Хмельницький національний університет
Факультет програмування
та комп'ютерних і телекомунікаційних систем
Кафедра комп'ютерної інженерії та системного програмування

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Мультипроцесорна система загального призначення на основі топології «кільце»
Назва теми

КвРКІ.170154.17.01.21 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія»*
Назва

Виконав: студент IV курсу, група КІ–17–1 _____ С.І. Талапчук
Підпис / Ініціали, прізвище

Керівник _____ О.М. Березький
Підпис, дата / Ініціали, прізвище

Нормоконтролер _____ С.М. Лисенко
Підпис, дата / Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної
Інженерії та системного
Програмування _____ Т.О. Говорущенко
Підпис / Ініціали, прізвище

«____» червня 2021 р.

Хмельницький 2021

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Програмування та комп'ютерних і телекомунікаційних систем

Кафедра Комп'ютерної інженерії та системного програмування

Освітній рівень Бакалавр

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Зав. кафедри Г.О.Говорущенко

" 11 " 01 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Галапчук Сніжані Іванівні

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Мультипроцесорна система загального призначення на основі топології «кільце»

Керівник проекту (роботи) Березький О.М., д.т.н., проф.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 05.02.2021 р. № 11

2. Строк подання студентом проекту (роботи) на кафедру 07.06.2021 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Дослідження предметної області та постановка задачі

Проєктування мультипроцесорної системи загального призначення на основі топології «кільце»

Програмно-апаратна реалізація та тестування мультипроцесорної системи загального призначення на основі топології «кільце»

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Схема з'єднання системи

Структура модулів ПЗ

Скріни роботи системи

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прий
Нормоконтроль	Лисенко С.М., доцент кафедри КІСП		
Антиплагіат	Нічепорук А.О., старший викладач кафедри КІСП		

7. Дата видачі завдання «11» 01 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Прим
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	11.01.2021	викона
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2021	викон
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2021	викон
4	Робота над розділом 2 – Проєктування мультипроцесорної системи загального призначення на основі топології «кільце»	01.04.2021	викон
5	Робота над розділом 3 – Програмно-апаратна реалізація та тестування мультипроцесорної системи загального призначення на основі топології «кільце»	30.04.2021	викон
6	Оформлення пояснювальної записки згідно вимог	31.05.2021	викона
7	Попередній захист ВКР	02.06.2021	викона
8	Захист ВКР на засіданні ЕК	Червень 2021 року	викона

Студент _____ С.І. Талапчук
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ О.М. Березький
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 170154.17.01.21 ПЗ	Пояснювальна записка	85		
			Графічні матеріали			
2		КвРКІ 170154.17.01.21 Е8	Схема з'єднання системи	1		
3		КвРКІ 170154.17.01.21 Е8	Структура модулів ПЗ	1		
4		КвРКІ 170154.17.01.21 Е8	Скріни роботи системи	1		

Зм	Арк	№ докум	Підпис	Дата	КвРКІ 170154.17.01.21 ВП		
Розробив		Галапчук		07.06.21	Відомість проекту	Літера	Аркуш	Аркушів
Перевір.		Березький		07.06.21		У	1	1
Н.контр.		Лисенко				ХНУ, КІ-17-1		
Затв.		Говорущенко		07.06.21				

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Мультипроцесорна система загального призначення на основі топології «кільце»».

Автор роботи: Талапчук Сніжана Іванівна.

Керівник роботи: Березький Олег Миколайович.

Пояснювальна записка: 85 с., 29 рис., 10 табл., 5 дод., 50 джерел.

Графічна частина: 10 презентаційних слайдів.

МУЛЬТИПРОЦЕСОРНА СИСТЕМА, СИМЕТРИЧНА АРХІТЕКТУРА, SMP, ТОПОЛОГІЯ «КІЛЬЦЕ», OPENMP, MPI.

Метою роботи є розробка мультипроцесорної системи загального призначення на основі топології «кільце».

У цій роботі розроблена мультипроцесорна система загального призначення на основі топології «кільце». Розроблена система реалізована на основі застосування процесорів AMD EPYC 7402 та симетричної архітектури. Розроблена система реалізована засобами мови програмування C++ та C# із застосуванням технології OpenMP, що дозволяє ефективно розпаралелювати завдання на процесори із збільшенням продуктивності та зменшенням часу виконання.

Підпис студента  Дата

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ОС - операційна система

КС - комп’ютерна система

SMP - симетрична мультипроцесорна система

ASMP - асиметрична мультипроцесорна система

ПК - персональний комп’ютер

CPU - центральний процесор

ПЗ - програмне забезпечення

ВСТУП

З моменту створення перших персональних комп’ютерів (ПК) пройшло багато часу, й така дорога та дефіцитна річ стала загальнодоступною. Наразі практично всі галузі потребують використання обчислювальних машин. Проте найважливішим напрямком залишився той, для якого вони власне і створювались, а саме для здійснення великих за обсягом обчислень. На сьогодні, цей напрямок стрімко розвивається, це пов’язано із зміною організації наукових досліджень і впровадженням числового моделювання, що повністю замінює складні, дорогі та небезпечні експерименти.

Звичайно, для виконання завдань такого масштабу необхідні значні обчислювальні ресурси. Вирішенням цієї проблеми стала ідея паралельної обробки даних, що базується на розподілі обчислень на безліч процесорів. Такий підхід дозволяє збільшити продуктивність обчислювальних машин у декілька разів. Прикладом таких систем є суперкомп’ютери.

Сьогодні, мультипроцесорні системи, які раніше асоціювали лише з суперкомп’ютерами, міцно закріпили позиції у всьому діапазоні обчислювальних системи: персональні комп’ютери, робочі станції, суперкомп’ютери та кластерні системи. Це дозволило збільшити доступність мультипроцесорних технологій та підвищити актуальність їх освоєння, оскільки для таких систем необхідне застосування спеціальних технологій програмування для повноти використання високопродуктивних обчислювальних систем. Зазвичай це можливо досягнути розбиттям програми для її виконання на окремих процесорах.

Мультипроцесорні системи розробляють насамперед для виконання величезних обсягів обчислень. При чому, необхідно створити єдину програму, для виконання якої задіяні всі наявні ресурси системи. Паралельне виконання програм істотно прискорює розв’язування задач. Варто зазначити, що мультипроцесорні системи завжди ефективніші ніж застосування однопроцесорних станцій такої ж кількості, оскільки за допомогою системи управління завданнями легко забезпечити рівномірне навантаження на процесори.

Актуальність роботи полягає в застосуванні топології «кільце», для з’єднання процесорів в мультипроцесорній системі.

Метою кваліфікаційної роботи є розробка мультипроцесорної системи загального призначення на основі топології «кільце».

Досягнути поставлену мету можна розв’язанням наступних основних задач:

– необхідно провести дослідження вже наявних рішень, для визначення вимог та правил побудови мультипроцесорних систем;

– виконати дослідження наявних проблем розробки та шляхи їх вирішення;

– розробити метод створення мультипроцесорної системи;

– виконати проєктування та реалізацію мультипроцесорної системи загального призначення на основі топології «кільце».

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

До сьогодення, практично всі ПК мали в своїй архітектурі лише один процесор загального призначення, але прагнення до зниження вартості мікропроцесорів і підвищення загальної продуктивності ПК підштовхнуло розробників створити щось нове, що задовільнило б усі потреби користувачів. Новими для виробників комп'ютерів стали системи, що наразі ми відносим до класу мультипроцесорних (рис. 1.1). Мультипроцесорна система – це багатопроцесорна комп'ютерна система, в якій наявні декілька процесорів, що використовують спільний адресний простір [7, 12].

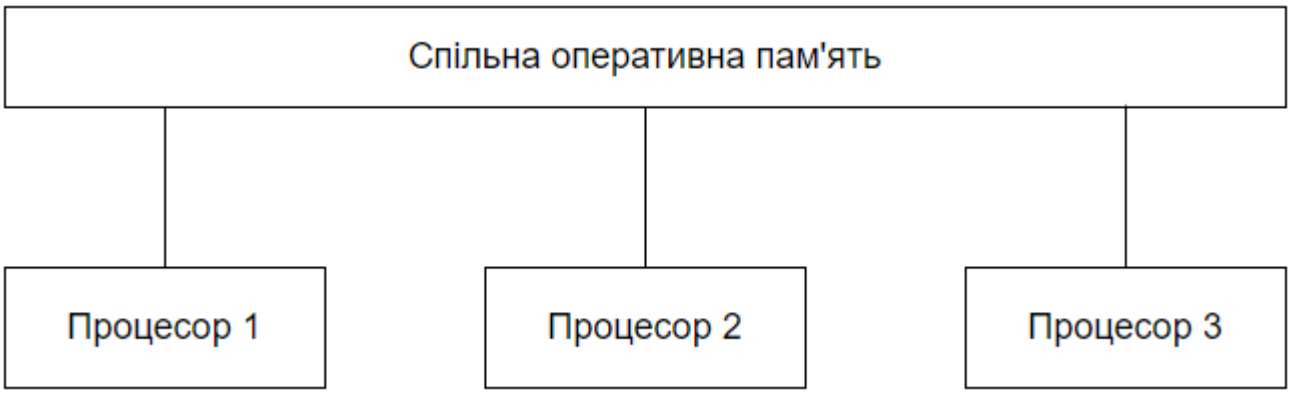


Рисунок 1.1 – Мультипроцесорна система із спільною пам'яттю

Характерними для мультипроцесорних систем стали наступні відмінні ознаки [2, 6, 19]:

1) наявні два або більше процесорів;

2) наявність доступу до загальної пам'яті, із збереженням однакового часу доступу до пам'яті;

3) наявність з'єднання, через системну магістраль або через будь який інший спосіб взаємодії;

4) наявність доступу до загальних засобів вводу-виводу (через роздільні канали або через один спільний);

5) здатність усіма процесорами виконувати одинакові функції;

6) наявність однієї загальної операційної системи для взаємодії між процесорами і програмами.

Обмін інформацією між компонентами у такій системі здійснюється на рівні окремих повідомлень, повних файлів або ж на рівні окремих елементів даних. Останній спосіб допомагає організувати більш тісну взаємодію між процесорами системи. Контроль розподілу процесів чи задач між процесорами, синхронізацію їхньої роботи та використання спільної оперативної пам'яті повністю лягає на операційну систему. Функції підтримки мультипроцесорних систем є в багатьох ОС, але зазвичай використовують саме Unix або подібні операційні системи [5, 9, 10].

Найбільш істотні переваги відносно однопроцесорних систем є підвищення продуктивності та надійності. Підвищення продуктивності зумовлена розподілом задач між процесорами, що дозволяє пришвидшити їхнє виконання. Надійність системи забезпечує наявність багатьох однотипних процесорів, що здатні виконувати ті самі задачі у випадку відмови одного із них. Отже, втрата одного процесора не призведе до виходу з ладу усієї системи чи її частини. Також, ще однією перевагою мультипроцесорних систем є можливість функціонального нарощування. Користувач у разі потреби може включати до її складу додаткові процесори. Це допоможе зробити ПК потужнішим.

Головною перевагою мультипроцесорних систем для виробників стала можливість виробництва схожих систем із різною продуктивністю. Сьогодні виробник може запропонувати покупцю безліч систем з однаковою чи схожою архітектурою, але з різною продуктивністю, що відрізняється кількістю процесорів і відповідно вартістю. Але слід враховувати, що більшість переваг є потенційними і не завжди вдається їх реалізувати, як наприклад нарощування.

Звичайно в таких системах є й низка вагомих недоліків, такі як:

– відносно маленька кількість процесорів, що можливо включити до складу таких комунікаційних систем;

– важка реалізація масштабованості системи;

– через використання спільної пам’яті, система має меншу продуктивність порівняно із системами з роздільною пам’яттю;

– порівняно з аналогічними системами з роздільною пам’яттю, має відносно високу вартість.

Зважаючи на це, об’єднати 30 і більше процесорів в одну систему досить важко, оскільки кожний процесор повинен мати доступ до блоків оперативної пам’яті фізично. Важко організувати й необхідну швидкість для такої системи, в 500 Мбайт/с і вище, вона обмежується відстанню від кожного блоку пам’яті до кожного процесора [3, 28, 33]. Тому такі системи, зазвичай, виглядають досить компактно і розміщуються в одному корпусі.

Використання спільної пам’яті, також передбачає для процесорів одночасний доступ до одного блоку пам’яті, для запису та зчитування даних. Такі функції можливі лише при реалізації відповідної кількості точок входу до кожного блоку пам’яті, яка дорівнює загальній кількості процесорів у системі. Для вирішення цієї проблеми створили низку варіантів побудови мультипроцесорних систем, що базуються на різних мережах. Чудовим прикладом є мережа типу «метелик» (рис. 1.2) [15, 20, 34].

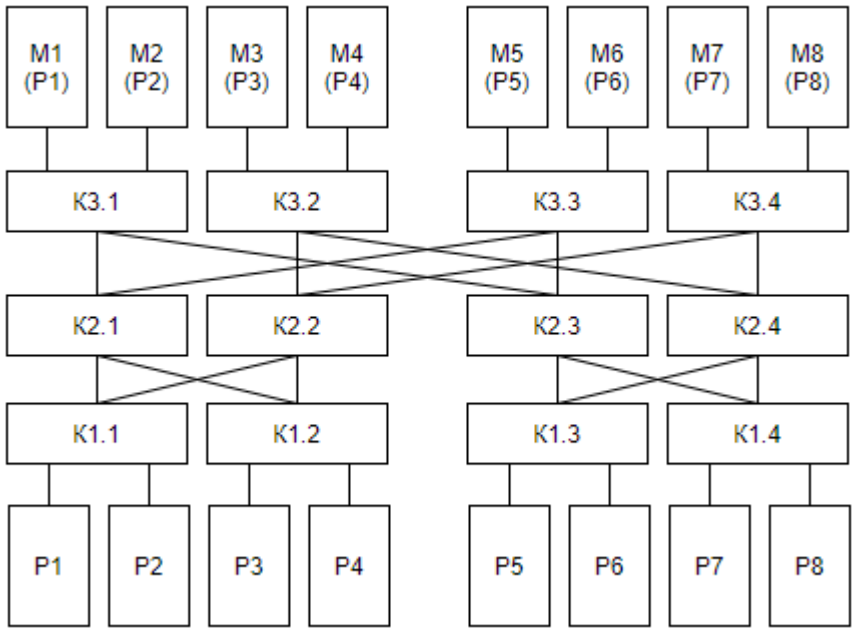


Рисунок 1.2 – Комунікаційна мережа типу «метелик»

На схемі зображений доступ восьми процесорів до блоків пам’яті через систему комунікаторів, але у відповідний момент часу до кожного блоку пам’яті має доступ лише один процесор, що значно сповільнює роботу усієї системи.

Мультипроцесорні системи, що відносяться до шинної архітектури складаються з процесорів, які через шину приєднанні до модулів пам’яті [30, 43, 46]. Найпростіша конфігурація мультипроцесорної системи містить в собі плату із шиною або материнську плату, у яку розміщують модулі пам’яті і процесори. Проблемою такої системи є перевантаженість та зниження продуктивності у разі збільшення кількості процесорів. Для вирішення цієї проблеми розробники розмістили між шиною та платою високошвидкісну кеш-пам'ять (рис. 1.3).

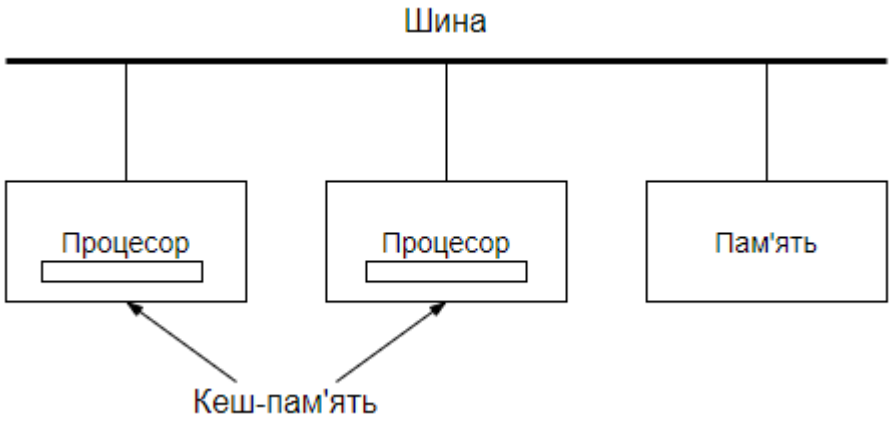


Рисунок 1.3 – Оптимізована архітектура КС

Дані, до яких процесори звертаються найчастіше, розміщуються в кеші, а отже при запиті, процесор звертається спершу в кеш-пам'ять, якщо необхідні йому дані не найдені, то лише тоді процесор звертається через шину до оперативної пам’яті. Звичайно, використання кеша тягне за собою значні проблеми, такі як неузгодження даних. Наприклад, перший процесор часто використовує деякий блок даних, тому зберіг його в своєму кеші, і при наступних запитах звертається лише до кешу. В той час, як другий процесор звернувся до пам’яті та змінив ці дані, в результаті пам'ять в системі стала неузгодженою й програмування і керування системою значно ускладнилося. Цей підхід дозволяє збільшити кількість процесорів за рахунок зменшення шинного трафіку, але

постає необхідність виділення додаткової пам’яті для кешу, що зазвичай становить від 512 Кбайт до 1 Мбайт [37, 48, 50]. Проте навіть використання кеш-пам’яті не усуває обмеженості масштабування. Тому слід пам’ятати, що для створення мультипроцесорної системи з більше ніж 256 процесорами, необхідно використовувати інші підходи.

Сучасні мультипроцесорні системи інколи будують за принципом кластерів. Для них характерне розбиття процесорів й оперативної пам’яті на кілька груп кластерів, що дозволяє отримувати швидкий доступ до оперативної пам’яті безпосередньо всередині кластера. Проте швидкість доступу процесора з одного кластера, до пам’яті з іншого кластера, значно зростає. Такі системи не вирішують конфлікти одночасного запису процесорами однакових даних та мають відносно велику вартість.

1.2 Аналіз наявного програмного та апаратного забезпечення предметної області

Незважаючи на те, що мультипроцесорні системи досить подібні і складаються з декількох процесорів, проте є багато варіантів для вибору пристроїв та об’єднання їх в систему. Відносно чого їх поділяють на симетричні та асиметричні мультипроцесори.

Симетричні мультипроцесори – це системи, в яких всі процесори відіграють однакову роль і мають ідентичний доступ до спільної пам’яті і периферійних пристроїв. Їх ще називають SMP-системи (Symmetric Multiprocessor) [1, 18, 40]. Симетричні мультипроцесорні системи можна розпізнати за деякими характеристиками:

1) декілька процесорів з однакової або схожою продуктивністю;
2) спільна пам'ять та адресний простір;
3) з’єднання процесорів за допомогою шини або будь-яким іншим способом;
4) доступ до загальних пристроїв вводу/виводу;

5) виконання процесорами однакових функцій (звідси і походить назва «симетричні»).

У SMP – системах можлива взаємодія між процесорами на рівні окремих елементів даних, що тісніше зв'язує процесори між собою. Незважаючи на те, що системи виступають як цілком симетричні, проте є недолік, що змушує засумніватись в цьому. При безпосередньому завантаженні системи один із процесорів стає головним (master). Це реалізовано для того, щоб користувач знав, який із процесорів керує завантаженням ОС по замовчуванню.

Побачити усі переваги симетричних мультипроцесорних систем можна порівнявши їх із однопроцесорними. Головною перевагою SMP є продуктивність [4, 8, 11]. Для її збільшення ОС розбиває задачу на частини та розподіляє між процесорами, це дозволяє пришвидшити виконання задачі шляхом паралельного виконання. Звичайно, в однопроцесорних системах таке виконання не можливе (рис. 1.4).

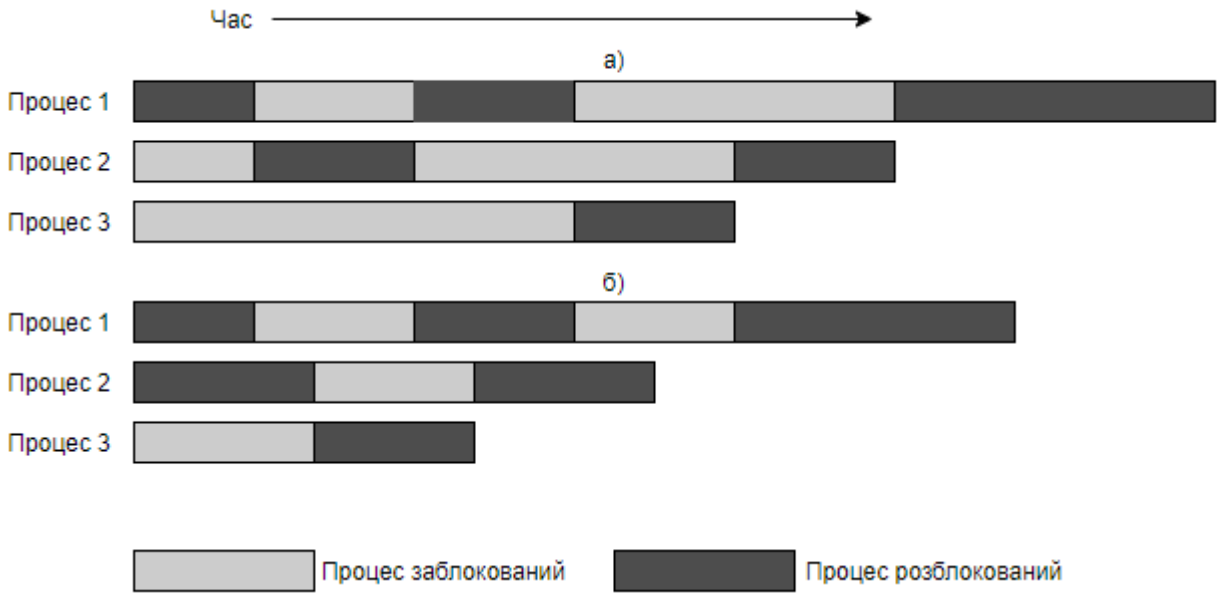


Рисунок 1.4 – Однопроцесорна і мультипроцесорна обробка
а – однопроцесорна; б - мультипроцесорна;

Наступною вагомою перевагою системи вважається готовність. Оскільки всі процесори здатні виконувати одні і ті ж функції, то відмова одного із компонентів не призводить до відмови системи. На відмінно від однопроцесорних систем.

Також до переваг слід віднести масштабованість системи. Завдяки цьому можливо створювати системи різної вартості та продуктивності. Звичайно усе це можливо лише за умови, що в операційній системі наявні засоби для підтримки паралелізму.

Типові симетричні мультипроцесорні системи в найзагальнішому вигляді мають не складну будову (рис. 1.5) [13, 21, 26].

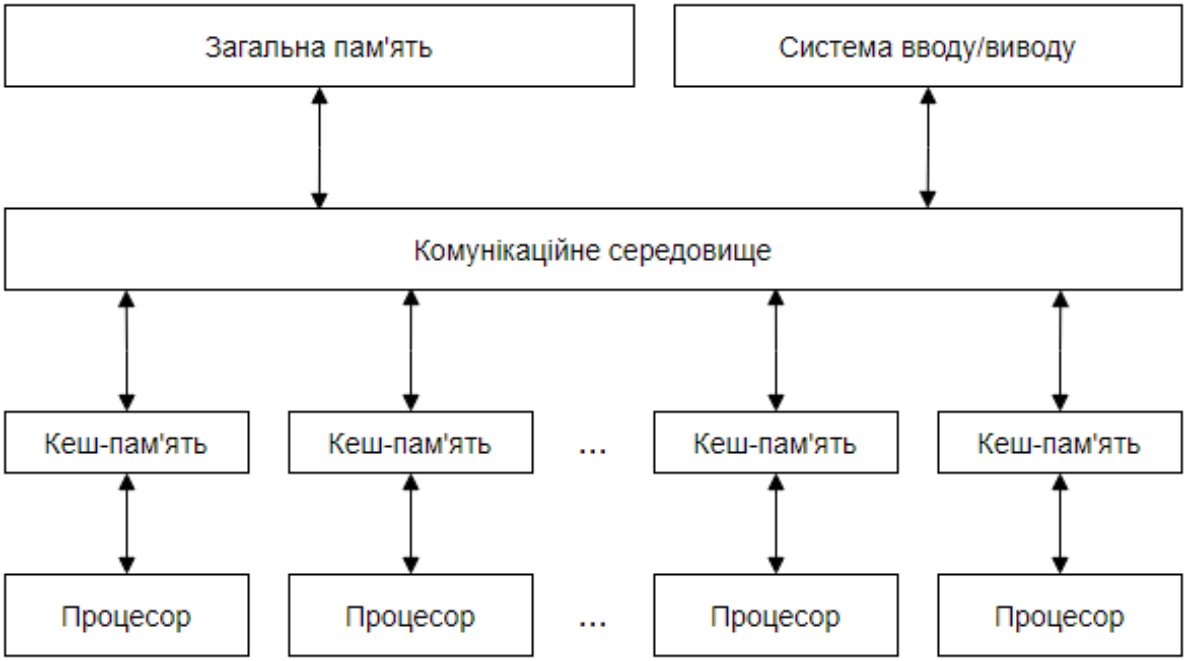


Рисунок 1.5 – Архітектура SMP

Зазвичай така система містить від 2 до 32 схожих або однакових процесорів, в якості яких часто використовують недорогі RISC-процесори або ж більш надійні CISC-процесори. Для зменшення навантаження на комунікаційне середовище, кожний процесор забезпечений кеш-пам’яттю першого і другого рівня. Для узгодження вмісту кешу процесорів застосовуються апаратні засоби. Також, для деяких симетричних мультипроцесорних систем характерне використання спільної кеш-пам’яті (рис. 1.6), що дозволяє вирішити проблему когерентності [14, 22, 23]. Проте таке рішення підходить лише для невеликої системи в якій число процесорів не перевищує чотирьох. В іншому випадку таке рішення призводить до зниження швидкодії кеш-пам’яті та підвищення вартості такої системи. Важливим в архітектурі SMP-систем є спосіб взаємодії процесорів з

пам’яттю та системами вводу/виводу. Наявні наступні види архітектури [24, 27, 29]:

– з багатопортовою пам’яттю;
– з комутатором;
– із загальною шиною;
– із пристроєм централізованого керування.

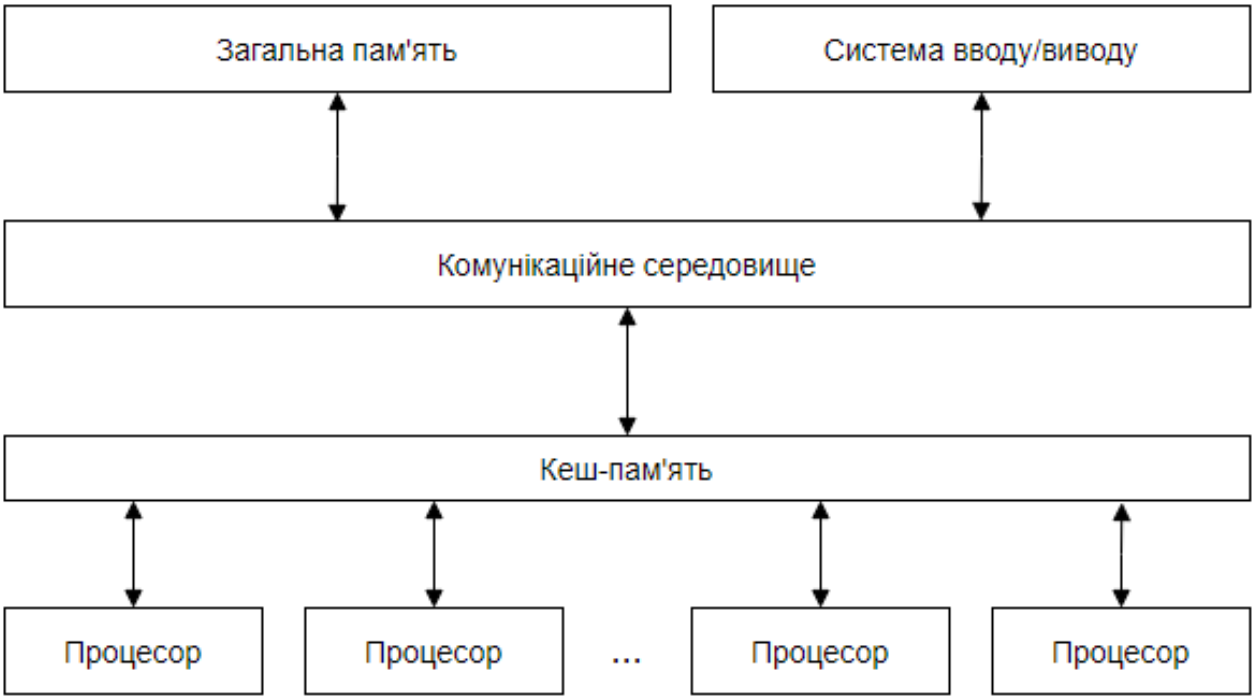


Рисунок 1.6 – Симетрична мультипроцесорна система із спільною кеш-пам’яттю

На відмінно від SMP-систем, в асиметричних мультипроцесорах усі процесори відіграють різні ролі або по-різному приєднані до периферичних пристроїв. Тобто кожному процесорові надається окрема функція чи задача. Для такої системи характерна наявність головного процесора (master processor), що планує роботу підлеглих процесорів (slave processors). Така архітектура зазвичай застосовується для дуже великих систем. Це дуже зручно, оскільки в випадку відмови головного процесора, один із підлеглих стає головним і виконує його функції, а в випадку відмови підлеглого – інший підлеглий процесор бере на себе його роботу. Асиметричну мультипроцесорну систему вважають простою, оскільки існує лише один процесор, що контролює усю роботу системи.

Прикладами таких систем є Burroughs B5000, CDC 6500, CDC 6700, DECsystem-1055 та PDP-11/74. У останньому прикладі система працювала під керуванням багатопроцесорної операційної системи RSX-11M. В PDP-11/74 усі процесори могли виконувати код та безпосередньо взаємодіяти з операційною системою. Також у ній передбачена можливість для всіх процесорів здійснювати введення-виведення, але доступ до периферійних пристроїв був не у всіх. Тому для організації роботи в такій системі використовували спеціальні запити. Це був головний недолік будови таких систем.

Порівняння SMP та ASMP подано у таблиці 1.1.

В ході порівняння, не можливо виокремити кращу будову мультипроцесорних систем, оскільки кожна із них використовується за своїх умов та потреб користувачів.

Симетричні мультипроцесорні системи найчастіше застосовується в промисловості, науці та бізнесі, де необхідне багатопоточне виконання. В той час як асиметричні мультипроцесорні системи застосовуються для високопродуктивних 3D-чипсетах в найсучасніших відеокартах [16, 17, 35]. Відповідно симетричні мультипроцесорні системи застосовуються значно частіше ніж ASMP-системи.

До мультипроцесорних систем відносять ще масово-паралельну архітектуру (MPP). Вона складається з мікропроцесорів, які поєднанні за допомогою мережі та мають власну локальну пам'ять (рис. 1.7) [25, 31, 32].

В загальній будові MPP-системи наявні декілька вузлів, що складаються з одного або кількох процесорів, з комунікаційного обладнання, з системи вводу/виводу та власної оперативної пам’яті. Тобто система має все необхідне для незалежного функціонування. При цьому на кожному з вузлів наявна або повноцінна ОС, або її урізаний варіант з підтримкою базових функцій ядра, за умови наявності керуючого комп’ютера з повноцінною операційною системою. Прикладом таких систем є RS/6000 SP2, Cray T3E та nCUBE2 [36, 38, 39]. Зазвичай в таких система з’єднання здійснене за допомогою високопродуктивного комутатора (рис. 1.8).

Таблиця 1.1 – Порівняння SMP та ASMP систем

Основа для порівняння	SMP-система	ASMP-система
Основні	Усі процесори виконують однакову роль.	Усі процесори відіграють різну роль у системі.
Процес	Процесор отримує завдання із загальної черги, або із черги для кожного процесора.	Головний процесор планує роботу для підлеглих процесорів.
Архітектура	Усі процесори мають схожу або однакову архітектуру.	Процесори можуть мати будь-яку архітектуру.
Зв'язок	Процесори спілкуються через спільну пам'ять.	Оскільки в системі наявний головний процесор, то спілкування не є необхідним.
Поламка процесора	У випадку поламки компонента системи, потужність системи знижується.	У випадку поламки головного процесора, його роботу виконує один із підлеглих. У випадку поламки підлеглого процесора, його завдання виконують інші процесори.
Легкість	Складний, оскільки процесори потребують синхронізації.	Простий, оскільки усім керує головний процесор.

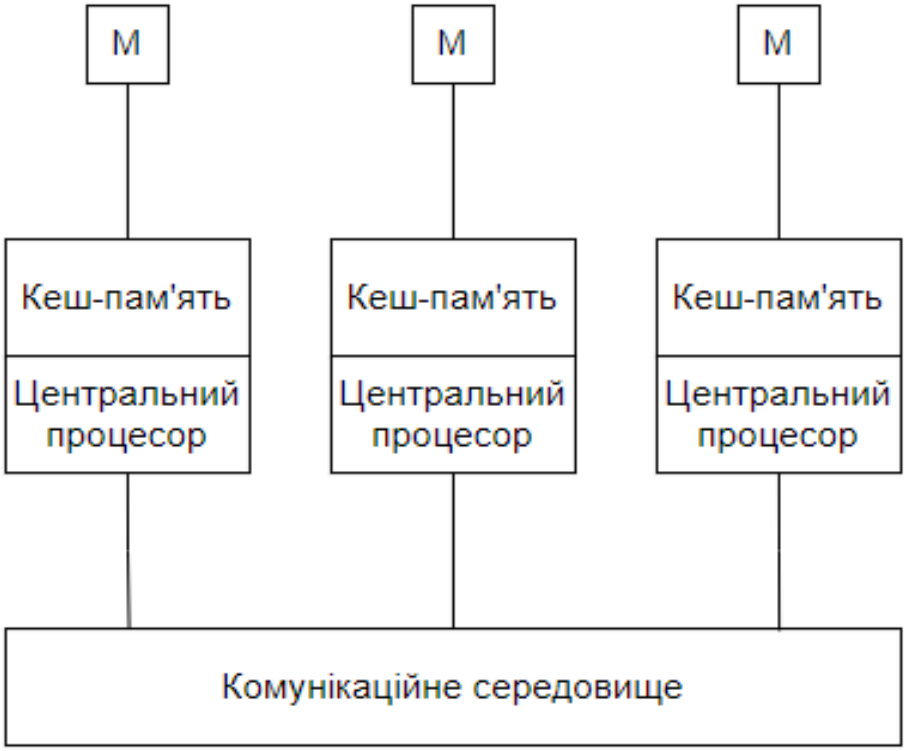


Рисунок 1.7 - Масово-паралельна архітектура (MPP)

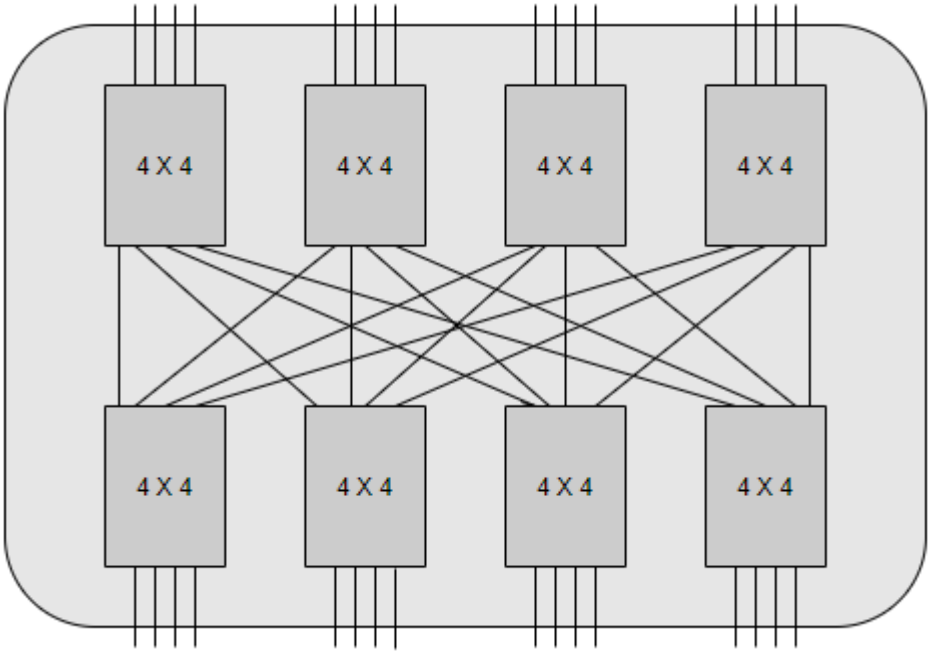


Рисунок 1.8 - Архітектура комутаційної плати 16x16 системи RS/6000 SP2

В MPP-системах прямий доступ процесори мають лише до власної пам’яті, але за допомогою механізму передачі повідомлень, вони можуть отримати необхідні їм дані з пам’яті іншого вузла. Така архітектура усуває відразу дві

проблеми: когерентність кеш-пам’яті та конфлікти при зверненні до пам’яті. Що дає змогу практично необмеженого нарощення числа процесорів в системі, збільшуючи при цьому її продуктивність. Головна властивість таких систем – масштабованість. Але з вирішенням одних проблем, виникли інші.

Головним недоліком на сьогодні являється проблема ефективності комунікаційного середовища. Наприклад, найпростішим та найефективнішим способом зібрати систему з 1000 вузлів це з’єднати їх між собою. Але для такого з’єднання необхідно 999 двонапрямлених комунікаційних каналів, що є не можливим. Для вирішення цієї проблеми застосовують мережеві топології. Для прикладу, в комп’ютерах Intel Paragon процесори утворили прямокутну двовимірну сітку, що знизило кількість комунікаційних каналів до чотирьох. У комп’ютерів Cray T3D /T3E використали всього шість комунікаційних каналів, застосувавши топологію тривимірного тора [41, 44, 49]. Проте найчастіше застосовують топологію n-мірного гіперкуба. Така архітектура характерна для комп’ютерів фірми nCUBE. Для обміну даними між процесорами, які не є сусідніми, у таких топологіях застосовується трансляція даних через проміжні вузли. Очевидно, що для такої передачі необхідні додаткові апаратні засоби. Тому, для з’єднання вузлів останнім часом все частіше застосовують високошвидкісні комутатори. Така топологія дозволяє процесорам обмінюватись даними без участі проміжних вузлів.

MPP-системи ідеально підходять для виконання незалежних програм. Оскільки кожна програма виконується на окремому процесорі і не впливає на виконання інших програм. Однак при розробці програм доводиться враховувати складну організацію пам’яті MPP-систем [42, 45, 47]. Вона має 3-х рівневу структуру:

– кеш-пам'ять;
– оперативна пам'ять вузла;
– оперативна пам'ять інших вузлів;

Зважаючи на це, обмін даними між вузлами виконується повільніше ніж обробка даних з власної оперативної пам’яті. Тому паралельні програми для таких

комп’ютерів написати значно важче, ніж для симетричних мультипроцесорних систем.

1.3 Визначення вимог до системи автоматизації та розробка технічного завдання

Будь-яка мультипроцесорна система є складним технологічним рішенням, здатним виконувати комплексні та ресурсоємні завдання з максимальною ефективністю. Варто наголосити, що подібні системи не є дешевими, тому перед розробкою будь-якої мультипроцесорної системи необхідно виконати аналіз вимог для уникнення лишніх витрат. Зазвичай такі системи мають ряд стандартних вимог:

1) ефективність – порівняно з однопроцесорними системами, мультипроцесорні мають непродуктивні витрати на синхронізацію та реалізацію паралельного виконання. Також, можливе виникнення проблеми оптимізації компонент, тому варто розрахувати потрібну потужність системи щоб уникнути простою процесорів;

2) гнучкість – це відповідність усім вимогам користувача, для зручного використання. Тобто, нефункціональні можливості: модульність, масштабованість, функціональна сумісність та переносимість.

3) стійкість до відмов – це можливість повноцінної роботи системи, при відмові одного чи більше компонент.

4) цілісність – так, як система складається з кількох процесорів, дуже важливим є забезпеченням повноцінного зв’язку між компонентами.

5) безпека системи – необхідно забезпечити захист від неправомірного доступу до системи. Важливими критеріями безпеки є:

a) конфіденційність множини користувачів для забезпечення доступу до пам’яті;

b) цілісність ресурсів і даних;

c) конфіденційність ресурсів і даних;

6) усунення проблеми когерентності кешу – поширена проблема великих мультипроцесорних систем із використанням суперечливих даних. Процесор бере необхідні та часто використовуванні дані із власного кешу, в той час, як інший процесор може змінити ці дані безпосередньо в операційній пам’яті. Звідси й виникає така проблема.

7) забезпечення безперебійного доступу до пам’яті – зважаючи на велику кількість процесорів, та одну загальну пам'ять, необхідно уникати виникнення проблеми одночасного доступу до пам’яті.

Варто зазначити, що суттєвою вимогою до мультипроцесорних систем є масштабованість. Для користувачів дуже важливо мати можливість збільшення потужності системи за потреби, але із мінімальними затратами. Проте реалізувати таку можливість досить важко.

Ще одним важливим критерієм є дотримання «принципу прозорості». Він полягає в приховувані непотрібних деталей реалізації для створення образу простої та ефективної системи. Принцип прозорості забезпечує цілісне сприйняття користувачем мультипроцесорної системи, при тому забезпечуючи усі необхідні знання для розуміння принципу роботи. Головними проявами принципу є прозорості:

1) доступу – це однаковий спосіб доступу до усіх об’єктів;

2) імен:

a) міграції – зміна розміщення об’єкта не впливає на зміну його імені;

b) розташування – доступ до даних здійснюється по імені, при цьому безпосереднє місцезнаходження ресурсів невідоме для користувачів;

3) реплікації (копіювання) – копії файлів виступають одним цілим і користувачу невідомо скільки їх існує, для забезпечення цього файлам даються одинакові імена;

4) одночасності доступу – можливість одночасного доступу до даних, без виникнення помилок ;

5) паралельності – можливість виконання процесів паралельно, без безпосередньої участі користувача;

6) розміру – можливість масштабування системи вшир та вглиб, без зміни бачення системи користувачем ;

7) продуктивності – для підвищення ефективності, мультипроцесорна система повинна мати можливість зміни конфігурації в залежності від навантаження.

Загалом вимоги та принципи прозорості взаємопов’язані. Так, наприклад, властивості розміру, імен і доступу забезпечують вимоги гнучкості, а копіювання та продуктивності – вимоги стійкості. В наступній таблиці розглянуто зв'язок між вимогами та принципами прозорості.

Таблиця 1.2 – Забезпечення вимог принципами прозорості

Вимоги	Гнучкість	Стійкість	Цілісність	Ефективність
Принципи прозорості	Розміру та імен	Розміру та копіювання	Доступу, продуктивності та копіювання	Паралельності, Продуктивності та одночасності доступу

В даній роботі, основною задачею є розробка мультипроцесорної системи загального призначення. Для забезпечення спілкування між процесорами, обчислювальні вузли в системі необхідно з’єднати між собою за допомогою топології «кільце». Сама система має бути побудована відповідно вимогам та принципам прозорості для забезпечення максимальної ефективності та комфорту для користувачів.

1.4 Висновки

Одним із найважливіших кроків розробки системи є аналіз наявних рішень. Розглядаючи та аналізуючи конкурентні системи можна підкреслити та запозичити переваги та покращити власну систему уникаючи наявних недоліків.

Звичайно, для розробки мультипроцесорної системи, спершу необхідно визначити деякі вимоги та принципи прозорості, яких варто дотримуватись. Такий підхід здатен значно скоротити час розробки та зменшити загальну вартість системи.

У даному розділі було досліджено предметну область та розглянуто наявні архітектурні рішення багатопроцесорних систем. Також вказано головні вимоги до системи та здійснено постановку задачі.

2 ПРОЄКТУВАННЯ МУЛЬТИПРОЦЕСОРНОЇ СИСТЕМИ ЗАГАЛЬНОГО ПРИЗНАЧЕННЯ НА ОСНОВІ ТОПОЛОГІЇ «КІЛЬЦЕ»

2.1 Вибір архітектури мультипроцесорної системи

Одним із головних кроків в розробці мультипроцесорної системи є вибір архітектури. На сьогодні, одними із найпопулярніших видів архітектури багатопроцесорних систем є симетрична та асиметрична. Кожна із них має свої переваги та недоліки. Тому, щоб обрати вдалу архітектуру, варто розглянути кожну окремо, та порівняти з потребами розроблюваної системи.

Загалом симетрична архітектура передбачає однорідність процесорів та ідентичність в підключенні до загальної схеми багатопроцесорної системи. Більша пам'ять в такій схемі розділяється між усіма процесорами, а для уникнення помилки доступу – застосовують власну кеш-пам'ять для кожного процесора, або одну загальну, що дозволяє уникнути проблеми невідповідності даних. Можливість нарощування кількості процесорів, так звана масштабованість, у SMP – системах обмежена, зважаючи на те, що всі процесори використовують загальну пам'ять (рис. 2.1).

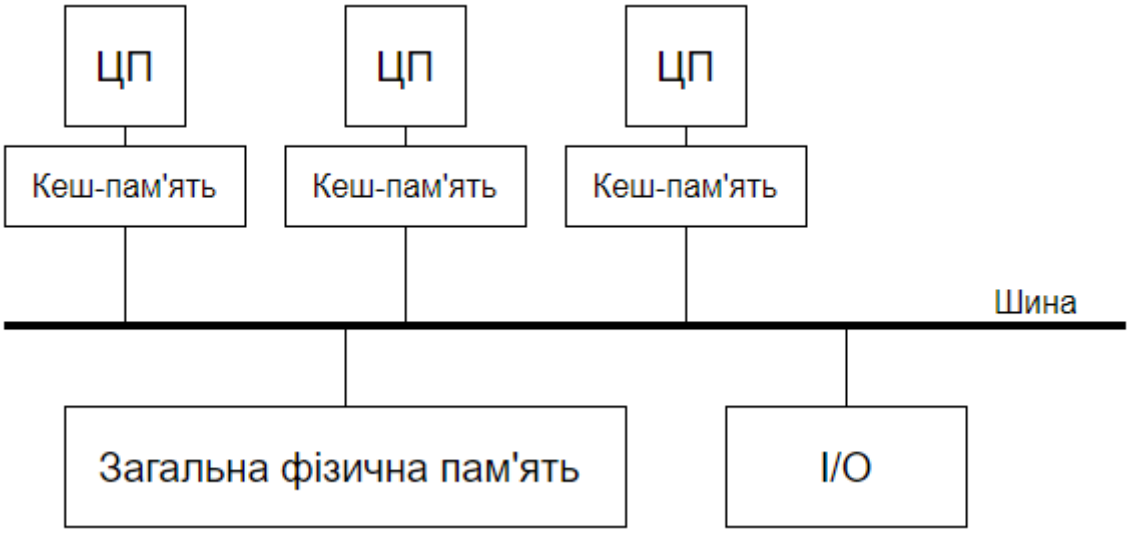


Рисунок 2.1 - Схематичний вигляд SMP - архітектури

Отже, розміщуватись усі процесори повинні в одному корпусі. Така система, її ще називають «масштабована по вертикалі», обмежує кількість процесорів до чотирьох чи восьми.

У симетричній архітектурі усі компоненти використовують загальну схему відображення пам’яті. Також, процесори можуть досить швидко обмінюватись даними, що забезпечує високу продуктивність системи.

На відмінно від симетричної архітектури, в асиметричній усі процесори можуть відрізнятись за характеристиками та функціональними ролями у системі. Наприклад, один процесор відповідає за керування ОС, другий призначений для обчислень, а третій для керування периферійних пристроїв і т.д.

Функціональна неоднорідність спричиняє у деяких фрагментах системи структурні відмінності. Наприклад, можуть відрізнятись підключенням до шини, способом взаємодії процесорів з пристроями та схемою підключення до них (рис. 2.2).

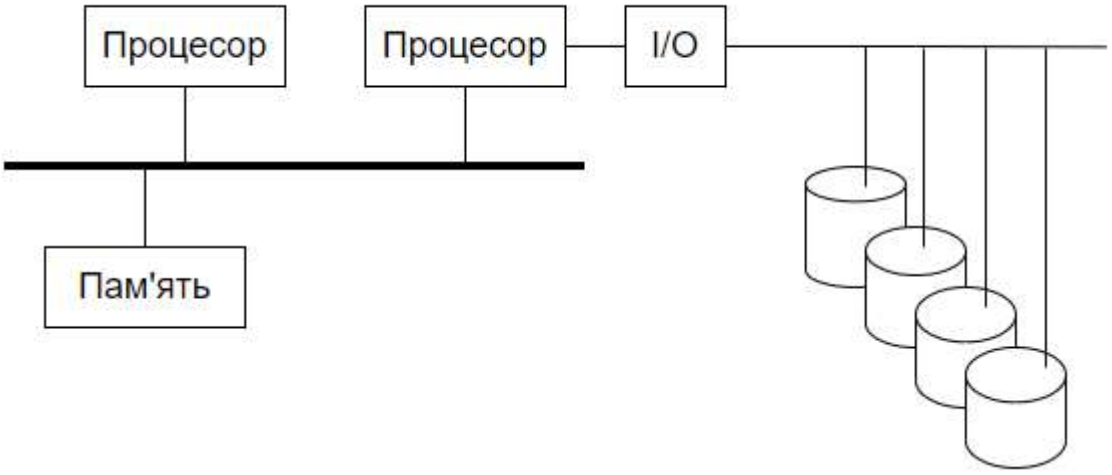


Рисунок 2.2 - Схематичний вигляд ASMP - архітектури

В асиметричній архітектурі реалізація масштабування відрізняється від симетричної. Оскільки відсутня вимога використання одного корпусу, система може складатись з декількох компонентів, кожний з яких може включати в себе від одного і більше процесорів. Таке масштабування називають «масштабування

по горизонталі». Варто зазначити, що кожний такий компонент називають кластером, а саму мультипроцесорну системи – кластерною.

Звичайно, від вибору архітектури системи, також залежить спосіб організації обчислювального процесору (симетричний чи асиметричний). Тому варто також розглянути ще один важливий аспект мультипроцесорних систем.

Симетричний спосіб організації процесу може бути реалізований лише із застосуванням симетричної мультипроцесорної архітектури. Характерним для нього є наявність загальної операційної системи для всіх процесорів. Причому, всі процесори рівноправно керують обчислювальним процесом і можуть виконувати однакові завдання. Також у симетричній організації процесори можуть одночасно працювати з однаковими чи різними модулями загальної операційної системи. Звичайно для цього необхідно, щоб в ОС була наявна властивість реєнтерабельності.

Операційна система в симетричному способі організації цілком децентралізована, тому усі її модулі виконуються на будь-якому процесорі. Загалом така система працює так: при завершені виконання завдання, процесор передає керування безпосередньо планувальнику завдань, який обирає завдання, що буде виконуватись наступним. Ресурси в симетричному способі не закріплюються за процесором, а виділяються в міру виникнення потреби. При такому підході забезпечується рівномірне навантаження на всі процесори. У випадну відмови компонента в симетричній системі, можливе досить просте реконфігурування.

Порівняно з симетричним способом, асиметричний може застосовуватись як в симетричній, так і в асиметричній архітектурі і залежить лише від типу ОС. Асиметричний спосіб організації процесу вважається найбільш простим. Він припускає наявність «головного» процесору, що відповідає за роботу ОС і управління «підлеглими» процесорами. Тобто процесор розподіляє завдання і ресурси між «підлеглими», що виступають в системі лише як пристрої для обробки і жодних дії по організації роботи не виконують. Це пов’язано з тим, що функції керування ОС повністю централізовані.

В асиметричних системах для ролі «головного» процесора зазвичай обирають найбільш продуктивний і надійний процесор. Звичайно такий підхід значно знижує надійність системи в цілому, адже процесори не завжди є взаємозамінними.

Отже, розглянувши можливі архітектури мультипроцесорних систем й порівнявши наявні переваги та недоліки, можна дійти висновку, що симетрична архітектура є найкращим варіантом для розроблюваної системи. Оскільки одним із найголовніших критеріїв системи є надійність. Також для системи важливо щоб процесори були рівні за функціональними ролями та могли досить швидко обмінюватись даними між собою. Все це можливо із застосуванням симетричної архітектури та симетричного способу організацій обчислюваного процесу.

2.2 Вибір методу розподілення доступу до спільної пам’яті

Симетричні мультипроцесорні системи мають значно менше недоліків ніж системи з розділеною пам’яттю. Проте головною проблемою в розробці SMP-системи є складність забезпечення комунікації між процесорами та пам’яттю й важкість реалізації одночасного доступу до спільних пристроїв. Існує декілька методів вирішення проблеми:

1) застосування загальної шини

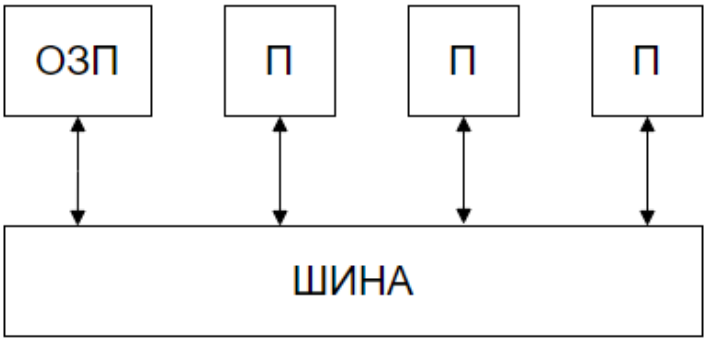


Рисунок 2.3 - Шинна модель комунікації

В даній моделі застосовується одна загальна шина до якої під’єднанні усі процесори та оперативна пам'ять. Шина виступає у ролі набору провідників, та обмежую доступ до пам’яті до одного процесора. Коли в компонента виникає потреба в отриманні доступу до пам’яті, він спершу намагається монополізувати доступ до шини. Причому інші процесори в той час не можуть взаємодіяти з пам’яттю і змушені чекати, коли звільниться доступ до загальної шини. Така модель не є доцільною для великих систем, у ній можливо ефективно об’єднати максимум 4-5 процесорів.

2) застосування системи матричних комунікацій

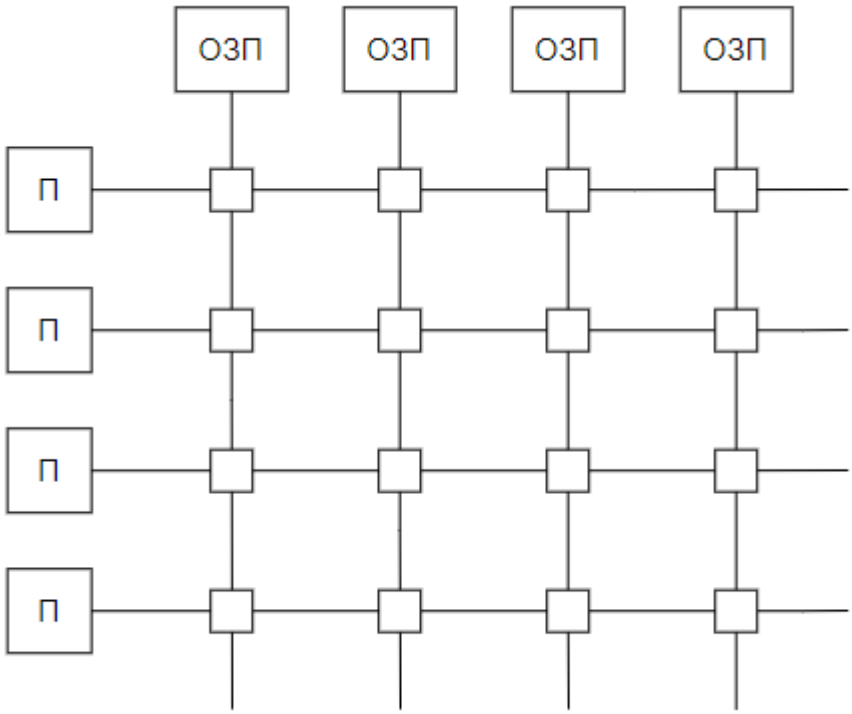


Рисунок 2.4 - Система матричних комунікацій

Підхід матричних комунікацій базується на кількох окремих блоках, які здатні працювати з процесорами, проте одночасно лише з одним. Для цього, на вузлах створеної сітки розміщують комутатор, який здатний дозволяти чи забороняти процесору звертатись до деякого блоку пам’яті. Необхідну кількість комутаторів для системи розраховують за формулою 2.1.

$$N = n^2, \qquad (2.1)$$

де n – кількість процесів.

Відповідно, при включені в систему 25 процесорів необхідно використати 625 комутаторів. Зважаючи на це, з’єднання великої кількості процесорів реалізувати практично неможливо. Також, недоліком такого з’єднання є досить висока вартість.

3) застосування омега мережі

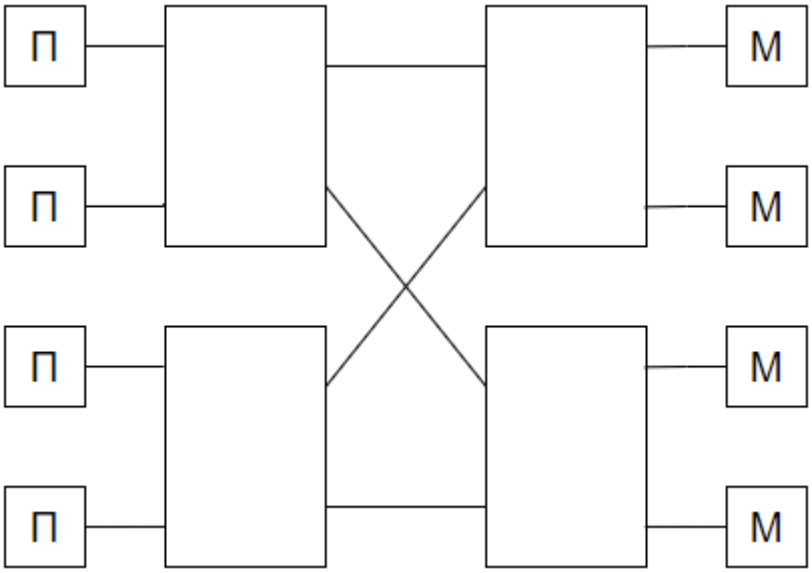


Рисунок 2.5 - Омега мережа

В омега мережі кожний процесор з’єднаний з комутатором, причому кожний його вхід може бути з’єднаний з будь-яким виходом. Зв'язок процесорів з блоками пам’яті здійснюється через один чи декілька комутаторів. Очевидно, що при такому сполученні кожний процесор має доступ до будь-якого блоку пам’яті. Перевагою такого з’єднання відносно попереднього є зменшення кількості комутаторів. Головний недолік омега мереж полягає в тому, що запит процесора до пам’яті чи навпаки, проходить через деяку кількість комутаторів, що призводить до великої затримки роботи системи. Тому, для забезпечення повноцінної роботи, без втрати швидкості, необхідно обирати комутатори з

високою швидкодією, що не є дешевим, а отже і сама система не буде загальнодоступною.

Таким чином, розглянувши можливі варіанти розділення доступу до спільної пам’яті та співставивши з потребами розроблюваної системи, можна дійти висновку що найкращим варіантом з’єднання є шинний. Оскільки в розроблюваній системі кількість процесорів не перевищує 5 та вартість системи повинна бути порівняно низькою.

2.3 Порівняння та вибір апаратного забезпечення

При проектуванні мультипроцесорної системи варто зважати на основні аспекти, такі як відповідність вимогам та принципам. Залежно від них, обираються необхідні компоненти, що в сукупності дозволить створити потужну та надійну систему. Розроблена система повинна відповідати таким вимогам:

- надійність;
- ефективність;
- легкість використання;
- швидкість роботи;
- гнучкість;
- масштабованість;
- цілісність;
- безпека системи.

Звичайно усе це можливо лише у випадку правильно обраних компонентів системи: процесорів, загальної шини та кеш-пам’яті.

Процесор – це компонент системи, що відповідає за управління процесом, обробкою даних і керуванням периферійними пристроями. Потужність та результативність роботи системи безпосередньо пов’язана зі швидкістю обробки завдань. Швидкий центральний процесор (CPU) дозволяє в декілька разів пришвидшити виконання завдань будь-якої важкості, але потужний процесор може і погіршити роботу, якщо обрати його не правильно. Головними критеріями, за якими відрізняють процесори один від одного, вважають розрядність, тактову

частоту та об’єм пам’яті. Важливою характеристикою є кеш-пам'ять. Вона потрібна для тимчасового зберігання даних, що використовує процесор при виконанні завдань. Наявність власного кешу значно зменшує час доступу до ОП. Ось чому варто порівнювати декілька варіантів процесорів та обирати найбільш придатний.

Для порівняння обрано п’ять процесорів: AMD EPYC 7402, AMD EPYC 7742, Intel Xeon Platinum 8280, Intel Xeon Gold 6248R, AMD EPYC 7763 (таблиця 2.3.1).

Таблиця 2.1 – Порівняння процесорів

Основа для порівняння	AMD EPYC 7402	AMD EPYC 7742	Intel Xeon Platinum 8280	Intel Xeon Gold 6248R	AMD EPYC 7763
Рік виходу	2019	2019	2019	2020	2021
Тип процесора	Серверний	Серверний	Серверний	Серверний	Серверний
Socket	Socket SP3	Socket SP3	LGA3647	LGA3647	Socket SP3
Кількість ядер	24	64	28	24	64
Кількість потоків	48	128	56	48	128
Базова частота	2800 MHz	2250 MHz	2700 MHz	3000 MHz	2450 MHz
Авторозгон	до 3350 MHz	до 3400 MHz	до 4000 MHz	до 4000 MHz	до 3500 MHz
Кеш L1, КБ	24x32 + 24x32	64x32 + 64x32	28x32 + 28x32	24x32 + 24x32	64x32 + 64x32
Кеш L2, КБ	24x512	64x512	28x1024	24x1024	64x512
Кеш L3, КБ	131072	262144	39424	36608	262144

Продовження таблиці 2.1 – Порівняння процесорів

Основа для порівняння	AMD EPYC 7402	AMD EPYC 7742	Intel Xeon Platinum 8280	Intel Xeon Gold 6248R	AMD EPYC 7763
Ядро	Rome	Rome	Cascade Lake-SP	Cascade Lake-SP	Milan
Техпроцес	7 nm	7 nm	14 nm	14 nm	7 nm
TDP	180 W	225 W	205 W	205 W	280 W
макс. температура	-	-	84° C	75° C	-
Шина	-	-	10.4 GT/s UPI	10.4 GT/s UPI	-
Контролер пам’яті	8 – канальний (DDR4-3200)	8 – канальний (DDR4-3200)	6 – канальний (DDR4-2933)	6 – канальний (DDR4-2933)	8 – канальний (DDR4-3200)
Контролер PCIe	PCI Express 4.0 (128 ліній)	PCI Express 4.0 (128 ліній)	PCI Express 3.0 (48 ліній)	PCI Express 3.0 (48 ліній)	PCI Express 4.0 (128 ліній)
Вбудовані модулі	Secure processor				GPIO, Шина LPC, I2C controller, Контроллер SATA 3.0, Secure processor, Контроллер SPI, Контроллер SD, Контроллер USB

Продовження таблиці 2.1 – Порівняння процесорів

Основа для порівняння	AMD EPYC 7402	AMD EPYC 7742	Intel Xeon Platinum 8280	Intel Xeon Gold 6248R	AMD EPYC 7763
Підтримка інструкції та технологій	• SSE • MMX • SSE4 • SSE3 • SSE2 • SSSE3 • AVX • SSE4A • AES • SHA • AVX 2 • BMI1 • AMD64 • F16C • FMA3 • EVP • SMEP • AMD-V • SMAP	• SSE2 • MMX • SSE • SSSE3 • SSE3 • SSE4 • AVX • SSE4A • AES • SHA • AVX 2 • FMA3 • BMI1 • F16C • EVP • AMD64 • SMAP • SMEP • AMD-V	• MMX • SSE • SSE2 • SSE3 • SSSE3 • SSE4 • AES • AVX • AVX 2.0 • AVX 512 • BMI1, BMI2 • F16C • FMA3 • EM64T • NX • VT-x • VT-d • Hyper-Threading	• MMX • SSE • SSE2 • SSE3 • SSSE3 • SSE4 • AES • AVX • AVX 2.0 • AVX 512 • BMI1, BMI2 • F16C • FMA3 • EM64T • NX • VT-x • VT-d • Hyper-Threading	• MMX • SSE2 • SSE • SSSE3 • SSE3 • SSE4 • AES • SSE4A • AVX • BMI1, BMI2 • AVX 2 • SHA • AMD64 • F16C • AMD-V • FMA3 • SMEP • EVP

Кінець таблиці 2.1 – Порівняння процесорів

Основа для порівняння	AMD EPYC 7402	AMD EPYC 7742	Intel Xeon Platinum 8280	Intel Xeon Gold 6248R	AMD EPYC 7763
Підтримка інструкції та технологій	• Precision Boost 2 • SMT	• Precision Boost 2 • SMT	• Turbo Boost • TXT • TSX • SMAP • SMEP	• Turbo Boost 2.0 • TXT • TSX • SMAP • SMEP	• Secure Encrypted Virtualization • Secure Memory Encryption • Precision Boost 2
Інші особливості	Підтримка ECC-пам'яті	Підтримка ECC-пам'яті	Підтримка ECC-пам'яті	Підтримка ECC-пам'яті	Підтримка ECC-пам'яті

Ще одним важливим аспектом порівняння є загальна швидкодія процесорів, вона подана на рисунку 2.6.

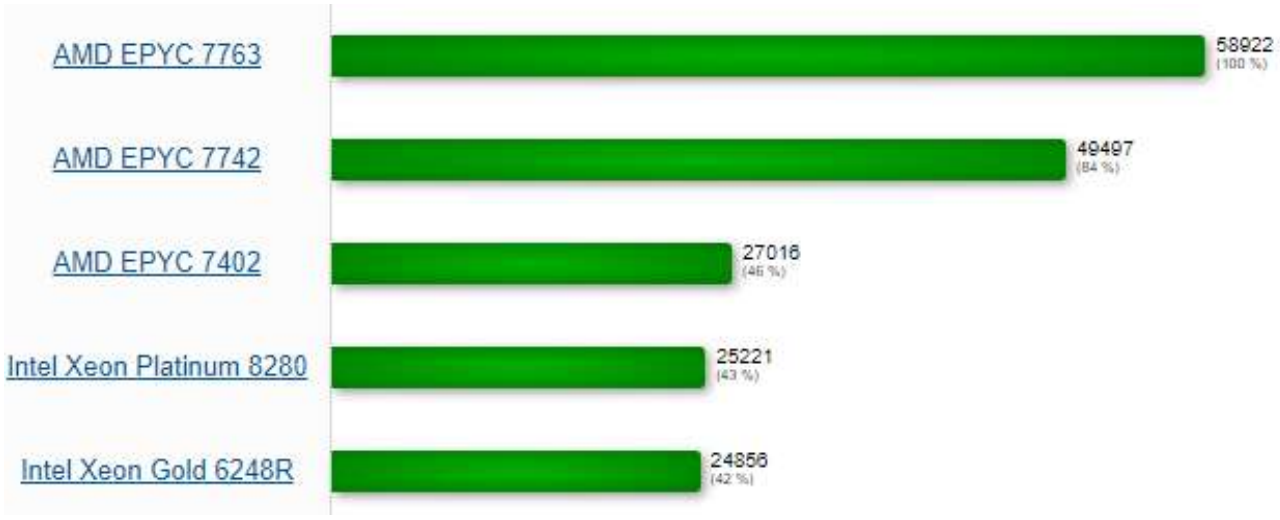


Рисунок 2.6 - Загальна швидкодія процесорів

Отже, на діаграмі можна побачити що процесор AMD EPYC 7763 має найбільшу швидкість роботи, що дає йому значну перевагу порівняно з процесорами компанії Intel. Проте, ще одним важливим критерієм вибору процесора є ціна (рис. 2.7).

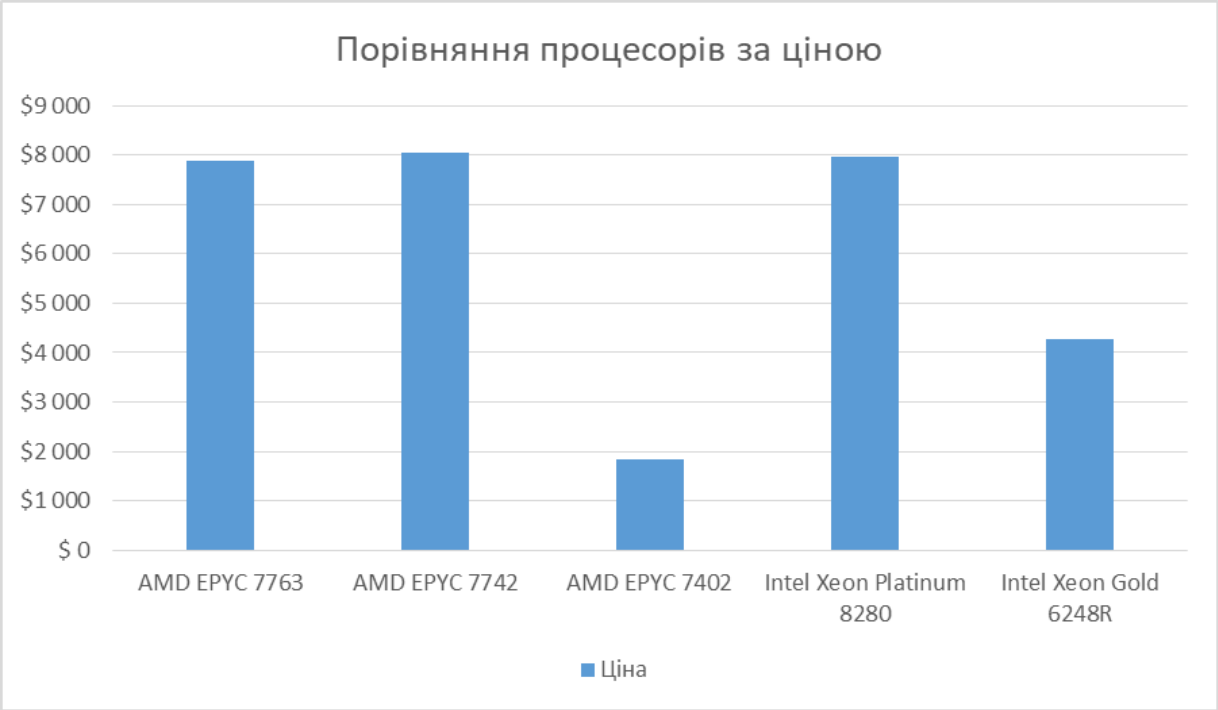


Рисунок 2.7 - Порівняння ціни процесорів

Судячи з діаграми найдешевшим процесором є AMD EPYC 7402. Звичайно, ціна не є запорукою якості та надійності процесора, проте й за характеристиками процесор не поступається іншим та має середню швидкодію, що можна побачити на рисунку 2.3.2. Також процесор має 24 ядра та 48 потоків, що цілком достатньо для нормальної роботи розроблюваної системи. В процесорі наявний великий обсяг кеш-пам’яті та висока тактова частота. Все це робить процесор AMD EPYC 7402, найкращим варіантом за співвідношенням ціна – потужність.

Проектуючи систему, варто пам’ятати що процесор лише один із головних компонентів. Звичайно, процесор – це мозок системи, проте навіть найпотужніший CPU не здатен працювати без шини, чи інших засобів комунікації.

Шина призначена для обміну інформацією між процесором та оперативною пам’яттю. Для забезпечення належної роботи системи, необхідно обрати шину, що здатна працювати з високими швидкостями. Типи і характеристики шин, що застосовують на сьогоднішній день подано в таблиці 2.2.

Таблиця 2.2. – Порівняння стандартних шин

Тип шини	Призначення шини	Розрядність шини	Тактова частота (МГц)	Пропускна здатність (Мб / с)
ISA	загальна	16	8	16
EISA	загальна	32	8	33
VLB (VESA)	графічна	32	40	130
VLB2	локальна	64	40	400
PCI	введення / виведення	32	33, 66	120, 133
SBUS	введення / виведення	32, 64	20, 25	80, 100
MBUS	процесор – пам’ять	64	50	125 (400)
XDBUS	процесор – пам’ять	64	50	310 (400)
AGP	локальна / графічна	32	133	533
PCI-X	загальна	64	133	1060

Звичайно у таблиці подано лише загальні характеристики шин, для кращого розуміння необхідно розглянути призначення кожного типу окремо:

– Системна шина ISA дозволяє паралельно передавати дані розміром 16 біт та звертатись до пам’яті розміром 16 Мбайт. ЇЇ використовують у сучасних комп’ютерах як шину введення/виведення, для зв’язку з повільними

периферійними пристроями. Проте така шина не є доцільною для використання в потужних комп'ютерах.

– Системна шина EISA забезпечує 32 – бітову передачу даних, має тактову частоту близько 8 МГц та адресний простір в 4 Гб. Також шина має швидкість обміну даними 33 Мбайт/с і сумісність з шиною ISA.

– Шина PCI використовується як локальна шина та шина введення/виведення. З'єднання шини процесора та PCI здійснюється з використанням перемички PC1 або контролера, що дозволяє різним процесорам працювати з PCI.

– Шина VME використовується, як шина введення/виведення в серверах та робочих станціях, що спроектовані на базі RISC-процесорів.

– Локальна шина AGP призначена виключно для графіки. ЇЇ використовують для розвантаження вузької (133 Мб/с) шини PCI при використанні відеоадаптера.

– Шина SBus має 32-бітову та 64-бітову розрядність, тактову частоту в 20 і 25 МГц і максимальну швидкість обміну даним в 80 або 100 Мбайт/с. В SBus передбачена можливість групової пересилки даних розміром до 128 байт. Шина працює в двох режимах: введення/виведення та прямого доступу до DVMA (віртуальної пам'яті), що дозволяє ефективно передавати великі обсяги даних.

– Шина MBus працює на частоті 50 МГц з мультиплексуванням адрес і даних. Розрядність MBus шини становить 64 біт, а загальний обсяг сигналів рівний 100. Також шина забезпечує когерентний стан кеш-пам'яті кількох процесорів. Головними властивостями системи вважають можливість масштабування, наявність високої пропускної здатності та підтримкою мультипроцесорної симетричної обробки. Така шина здатна нормально функціонувати лише у невеликих системах (до чотирьох процесорів).

– Шина XDBus виконує такі ж функції, що і шина MBus, але здатна з'єднувати велику кількість процесорів. Шина працює в режимі розділення транзакцій, що дозволяє при піковій продуктивності в 400 Мбайт/с, підтримувати максимальну в швидкість понад 310 Мбайт/с.

Отже, порівнявши системні шини, можна помітити, що для розроблюваної системи необхідно обрати шини для системи введення/виведення та для з'єднання процесорів з оперативною пам'яттю. Найкращими варіантами для багатопроцесорних систем є SBus, MBus і XDBus, причому в даному проекті необхідно використати шину SBus в якості шини введення/виведення та XDBus – для з'єднання декількох процесорів та пам'яті. Це дозволить процесорам швидко пересилати дані й нарощувати загальну потужність системи шляхом її масштабування.

2.4 Вибір ОС

Операційна система, ОС - базовий комплекс комп'ютерних програм, що забезпечує зв'язок з користувачем, управління апаратними засобами комп'ютера, введення/виведення даних, а також виконання прикладних програм та утиліт. Важливою властивістю є відсутність або наявність в ОС засобів підтримки багатопроцесорної обробки. Її наявність призводить до ускладнення всіх алгоритмів керування ресурсами. Приклади таких систем: Solaris 2-х фірми Sun, Open Server 3.x Santa Crus Operations, Windows NT/7/8/10 і NetWare 4.1.

Мультипроцесорні операційні системи класифікуються за способом організації обчислювального процесу на: асиметричні операційні системи та симетричні операційні системи.

Асиметрична ОС повністю виконується тільки на одному процесорі системи, при цьому розподіляє завдання між іншими процесорами. На відмінно від симетричної ОС, яка повністю децентралізована, що означає використання всього пулу процесорів, при цьому пул ділиться на системні та прикладні завдання.

Вищезазначена характеристика описує управління лише одним типом ресурсу - процесором. Специфіка мультипроцесорних операційних систем залежить також від інших ресурсів, таких як підсистеми керування пам'яттю, файлами, пристроями введення-виведення, та навіть реалізації мережевих інтерфейсів. Оскільки, при реалізації мережевих функцій виникають завдання,

пов’язані з розподіленим характером зберігання і обробки даних в мережі, тиражування даних їх узгодження та резервне копіювання.

Найчастіше при розробці мультипроцесорних систем застосовують UNIX-подібну операційну систему.

UNIX - це сімейство багатозадачних комп'ютерних операційних систем, які стали досить популярними та набули стрімкого поширення на машинах з різною потужністю обробки, від мікропроцесорів до великих ЕОМ, забезпечуючи на них загальні умови виконання програм.

UNIX-подібна система, зазвичай, ділиться на дві частини. Одну частину складають програми та сервісні функції, те, що робить операційне середовище UNIX таким популярним. Ця частина легко доступна користувачам, вона включає такі програми, як обмін повідомленнями, командний процесор, системи обробки початкового програмного коду і пакети обробки текстів. Інша частина включає в себе власне операційну систему, яка підтримує програми і функції.

Особливості UNIX-систем, що відрізняють дане сімейство від інших ОС:

– деревоподібна файлова система (реєстрозалежна);

– введення-виведення лише через файлові дескриптори;

– на рівні системних викликів, файл є потоком байт;

– команда є ім'ям файлу програми, спеціальна реєстрація та розробка, як у RSX-11 та RT-11 не потрібна;

– використання 2 рівнів привілеїв процесора замість 4 в VMS;

– відмова у використанні оверлеїв на користь поділу програми на декілька підпрограм, спілкування яких відбувається через конвеєри або тимчасові файли;

– повсякденні завдання обробки тексту супроводжуються використанням відповідних утиліт;

– використання файлів для зберігання налаштувань, на відміну від бази даних налаштувань, як в Windows.

Переваги UNIX-подібної системи для використання в багатопроцесорній конфігурації:

– мінімальний розмір ядра та підтримка найнеобхідніших служб;

– популярність - велике розповсюдження забезпечує наявність хорошої документації;

– простий доступ до файлів – будь-який файл є звичайною послідовністю байт, тому програми можуть реалізовувати власні методи зчитування, запису, структурування, не інформуючи про це операційну систему;

– висока ступінь переносимості - більша частина ядра системи написана на С, це дозволяє відносно легко адаптувати систему до різних апаратних платформ;

– простий, потужний та уніфікований інтерфейс вводу-виводу - представляючи доступні пристрої як файли, система дозволяє взаємодіяти з ними використовуючи доступні файлові команди;

– чітка файлова система, звернення до файлу без суперечностей та власний функціонал захисту даних.

Недоліки UNIX-подібної системи для використання багатопроцесорної конфігурації:

– надто велика конфігурованість - кожен бажаючий користувач має право вносити власні зміни в ядро, необдумані зміни збільшують ймовірність несумісності з іншими системами;

– необхідність ручного розширення для підтримки апаратних пристроїв та периферії. У випадку з UNIX, на вході чисте ядро, з мінімальною сумісністю пристроїв необхідно власноруч здійснювати розширення ядра, для забезпечення їх підтримки;

– необхідність у вирішенні проблеми безпеки цілісності даних;

– необхідність переробки існуючих у системі алгоритмів.

Головним недоліком таких систем є проблема цілісності даних, що спричинена паралельною роботою декількох процесорів.

В операційній системі UNIX є базовий механізм який вирішує цю проблему, а саме ядро не може вигрузити один процес та переключитись на контекст іншого, коли робота виконується в режимі ядра. Також, виконуючи критичний сектор програми обробник, що працює з винятками, приховує їх, оскільки це може пошкодити структуру даних на яких вони виникли.

Однак, в багатопроцесорній системі порушення може виникнути навіть незважаючи на вищезазначені захисні міри. Існує три способи вирішення цієї проблеми:

1) виконувати критичні операції на одному процесорі та покладатись на стандартні методи збереження однопроцесорної системи;

2) регламентувати доступи до критичних секторів та використовувати блокування;

3) переробка існуючих алгоритмів, з метою усунення конкуренції за використання структур.

Не зважаючи на деякі недоліки таких систем, UNIX-подібні ОС здатні чудово працювати в режимі багатозадачності, порівняно з ОС Windows, для використання яких необхідно безліч утиліт. Також, операційна система UNIX має високу ступінь переносимості та надійності. Головною перевагою ОС, відносно розробки даного проекту, є підтримка симетричної роботи процесорів, що полегшить реалізацію мультипроцесорної системи.

2.5 Порівняння та вибір середовища паралельного програмування

Проектуючи мультипроцесорну систему зазвичай вважають, що для керування роботою усієї системи та розпаралелювання задач на процесори, вистачить лише операційної системи. Проте, досить часто її виявляється недостатньо. Звичайно, операційна система якимось чином може розподіляти задачі по процесорах, але у такому випадку, переваги таких систем не завжди очевидні.

Зазвичай, для розпаралелювання задач використовують різні механізми, один із них – породження потоків (threads). У ньому створюються процеси, які не потребують окремого адресного простору, але які розподіляються між процесорами у мультипроцесорній системі. У мові програмування С такий підхід використовується за допомогою виклику системних функцій для розпаралелювання програми або в режимі задання компілятору відповідних директив, що відповідають за розпаралелювання програм.

Останнім часом, усі симетричні мультипроцесорні системи підтримують стандарт POSIX Pthread і включають компілятори для мов програмування, що найчастіше використовуються для розпаралелювання коду та програм. Зокрема, у таких компіляторах наявні спеціальні директиви, але вони досить обмеженні та несумісні між собою. Усе це змушує розпаралелювати програми для кожної платформи окремо.

Наразі, одними із найпопулярніших систем програмування є OpenMP та MPI. Вони є узагальненням та розширенням існуючих наборів директив. OpenMP реалізує ідею «інкрементального розпаралелювання» в моделі загальної пам'яті. У неї входять специфікації набору процедур, середовища та директив компілятора. Для застосування OpenMP не потрібно створювати нову програму, а в код існуючої послідовної програми додати необхідні директиви. При цьому OpenMP дає можливість розробникові контролювати поведінку розпаралелення задач (рис. 2.8).

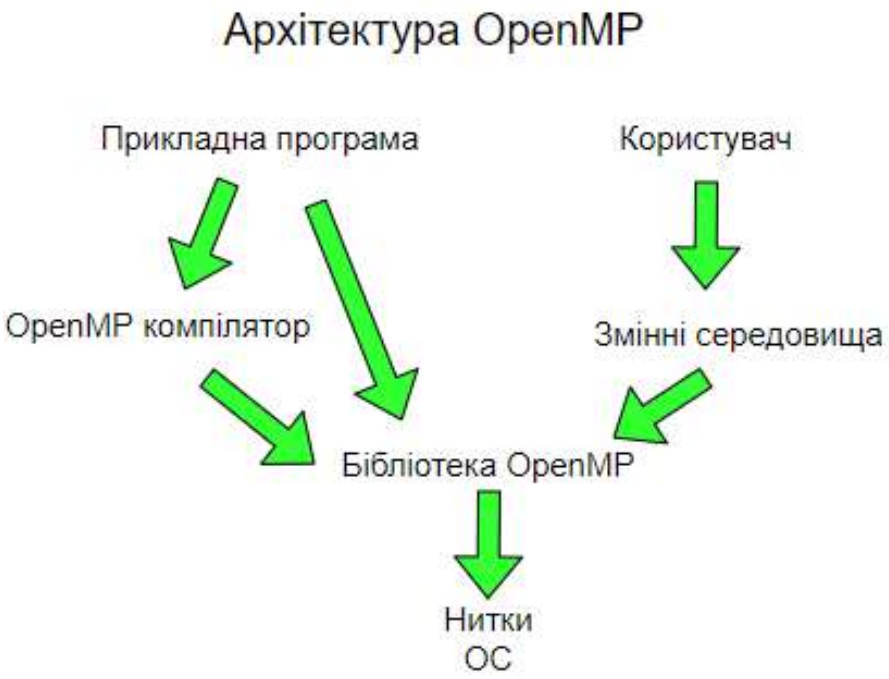


Рисунок 2.8 - Архітектура OpenMP

Така програма розділяється на послідовні та паралельні області. На головному потоці, що створюється на початку запуску програми, виконуються

послідовні області, а для паралельних – головний потік створює додаткові. Варто зауважити, що OpenMP здатна однаково чудово працювати на багатопроцесорних та однопроцесорних системах. При чому, в останньому випадку код розпаралелювання просто ігнорується.

На відмінно від OpenMP, MPI програми передбачають набір довільно фіксованого числа процесорів, що взаємодіють через бібліотеки MPI (рис. 2.9). Кількість процесорів, в такій програмі, визначається при ініціалізації і нумеруються в інтервалі від 0 до groupsize – 1. Взаємодії процесів в MPI відбуваються через повідомлення та поділяються на парні та групові. У парних необхідно вказувати дані та конверт, а у групових зазначати ще й комунікатори. Комунікатор - це подання групи процесів разом з контекстом. Контекст використовується для уникнення завад між повідомленнями, що незв'язані між собою.

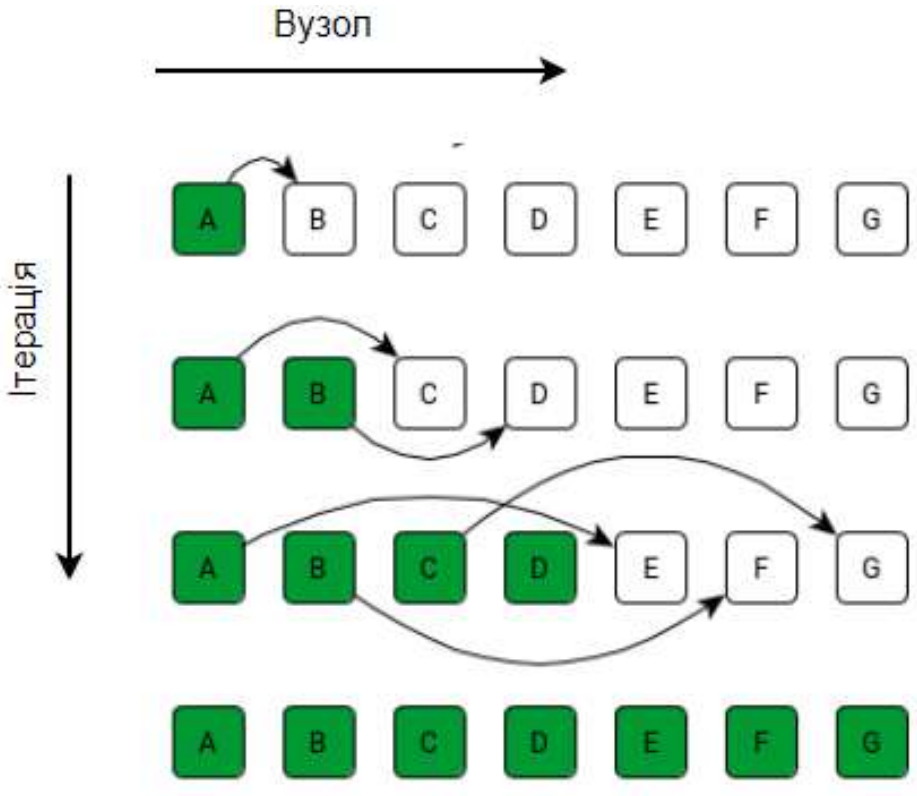


Рисунок 2.9 - Передача даних в MPI

Також, в MPI існують декілька режимів виконання:

– блокуючий (синхронний) – виклик функції не завершується до повного закінчення операції обміну;

– неблокуючий (асинхронний) – виклик функції не залежить від операції обміну і може завершитись до її закінчення;

– колективний – усі процеси виконують MPI – функції для взаємодії один з одним;

– локальний – для обміну не потрібно взаємодіяти з іншими процесорами.

Усі режими в MPI не є взаємовиключаючими, тому можливо використовувати в одному виклику одночасно декілька. Це дає перевагу порівняно з OpenMp.

Звичайно, для повноцінного порівняння систем програмування варто зважати на ще один критерій, а саме час виконання задачі. Для порівняння використано програму для пошуку входження слів, що працює з різним об’ємом даних та застосовує різні технології програмування, що дає змогу порівняти час затрачений на виконання кожної з них (таблиця 2.3).

Таблиця 2.3 – Середній час виконання програми

Розмір файлу (Mb)	Послідовний	OpenMP	MPI
25	10.066	6.047	5.809
50	22.230	13.503	12.341
100	49.509	31.837	27.362
200	105.532	67.674	69.764
400	217.931	136.348	140.528

У таблиці 2.3 можна побачити, що послідовне виконання програми затрачає часу вдвічі більше ніж технології OpenMp та MPI. Це пов’язано з тим, що при паралельному виконанні задіються усі наявні процесори, а при послідовному – лише один. В результаті відбувається велике навантаження на процесор, що спричиняє гальмування усієї системи. Наочно побачити різницю в часі можна

розглянувши графік порівняння часу виконання програми із використанням різних технологій (рис. 2.10).

Рисунок 2.10 - Середній час виконання програми із застосуванням різних технологій

Очевидно, що використання в програмі OpenMP та MPI робить систему більш продуктивною та дає значний виграш у часі. Проте у мультипроцесорній системі із використанням спільної пам’яті варто використати OpenMP, оскільки в ній процеси використовують спільний адресний простір, до якого вони звертаються із запитами на зчитування та запис даних. Головною перевагою OpenMP є відсутність монопольного володіння даними, що дозволяє уникати не потрібних обмінів даними між процесорами. При цьому, OpenMP є досить гнучким й надає користувачу контролювати поведінку програм та розпаралелювати роботу деяких областей коду не переписуючи його. Усе це значно спрощує розробку та керування мультипроцесорною системою.

2.6 Висновки

В даному розділі було визначено основні вимоги до системи та обрано найбільш вдалу архітектуру побудови мультипроцесорної системи. Також було обрано спосіб обчислювального процесу та метод розділення доступу до спільної пам’яті зважаючи на потреби системи й користувачів. Для вибору найкращих засобів, виконане порівняння операційних систем, апаратного та програмного забезпечення. Виконаний вибір програмних та апаратних засобів розробки мультипроцесорної системи.

Також, в розділі було здійснено оцінку швидкості роботи багатопроцесорної системи з використанням різної кількості процесорів та із застосуванням найбільш дієвих технологій паралельного програмування. В результаті чого, було обрано технологію, що задовольняє усі вимоги системи та значно покращує продуктивність мультипроцесорної системи.

3 ПРОГРАМНО-АПАРАТНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МУЛЬТИПРОЦЕСОРНОЇ СИСТЕМИ ЗАГАЛЬНОГО ПРИЗНАЧЕННЯ НА ОСНОВІ ТОПОЛОГІЇ «КІЛЬЦЕ»

3.1 Застосування топології «кільце», для з’єднання процесорів в системі

Одним із головних кроків в реалізації мультипроцесорної системи є безпосереднє з’єднання компонентів. Звичайно, для об’єднання процесорів можна використати лише програмне забезпечення чи загальну шину. Проте таке з’єднання буде значно сповільнювати систему. Для вирішення цієї проблеми застосовують топології.

Під топологією мультипроцесорної системи мають на увазі спосіб з’єднання процесорів між собою за допомогою комунікаційних каналів. Причому ефективність виконання задач та програм, з використанням розпаралелювання, й можливість масштабування таких систем, визначають за властивостями обраної топології. Варто розуміти, що для підвищення ефективності та надійності системи, необхідно, щоб фізична топологія розроблюваної системи та топологія задачі, для якої призначена система, були узгодженні.

Зважаючи на те, що розроблювана система складається з трьох процесорів та має загальне призначення, варто використати топологію «кільце» (рис. 3.1).

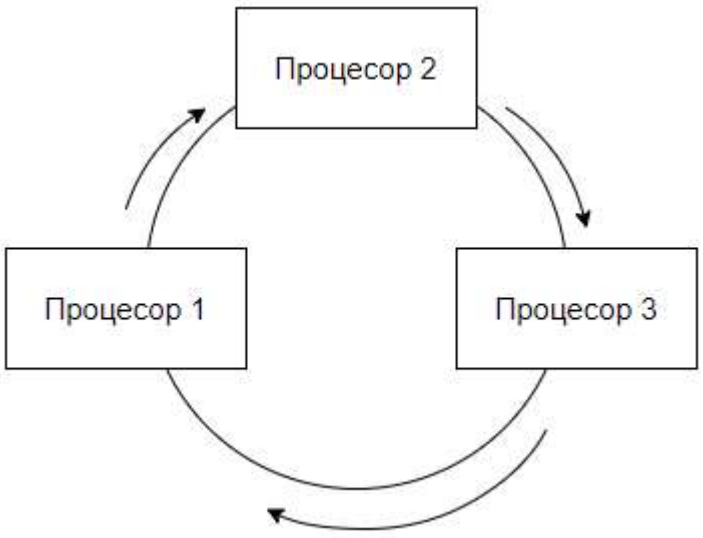


Рисунок 3.1 – Топологія «кільце»

В самому загальному вигляді кільцева топологія представляє собою процесори, кожний з яких з’єднаний з двома іншими. Причому останній та перший з’єднуються між собою утворюючи при цьому кільце. Сигнал в цій топології односторонній, тобто проходить від одного процесора до наступного, аж поки не досягне необхідного компонента. Для визначення вузла, що передає чи отримує по запиту деяку інформацію, застосований маркер. Під час пересилання інформації усі компоненти беруть активну участь, тому у разі поламки хоч одного, вся система перестає працювати.

У топології «кільце», як і в шинній, кількість компонентів може бути дуже великим. Оскільки відсутні конфлікти та центральний (головний) елемент, на який припадає найбільше навантаження, а отже така система має досить високу стійкість до навантажень, що дає змогу об’єднувати велику кількість процесорів. У топології застосовується зв'язок типу точка-точка, що дозволяє уникнути витрат на додаткові компоненти.

Звичайно, у топології «кільце» є свої переваги та недоліки.

До основних переваг топології відносять:

– простоту використання та установки;

– можливість уникнення використання додаткового обладнання;

– стійкість та надійність системи без втрати продуктивності при великих навантаженнях та обсягах даних;

– можливість масштабування;

– можливість застосування симетричної обробки даних.

Недоліки топології «кільце»:

– активна участь кожного компонента, навіть якщо процесор не задіяний безпосередньо до виконання завдання;

– необхідність вимикання системи при додаванні процесора;

– важкість виявлення несправності;

– важкість налаштування системи;

– необхідність мати додаткові входи для з’єднання елементів між собою;

– вихід з ладу хоч одного з кабелів з’єднання, унеможливлює роботу усієї системи.

Звичайно, більшість з наявних недоліків відносяться до систем із великою кількістю процесорів, таких як суперкомп’ютери, тому до розроблюваної системи, кількість процесорів якої становить всього три, їх віднести не можна.

Для розробки мультипроцесорної системи загального призначення використано три серверних процесора. Як зазначалось раніше в пункті 2.2, кожний процесор вже має в собі вбудовану кеш-пам'ять, тому з’єднати необхідно процесори між собою та загальну фізичну пам'ять із процесорами. Схему з’єднання компонентів показано на рисунку 3.2, на якій стрілкою вказано напрямок руху даних.

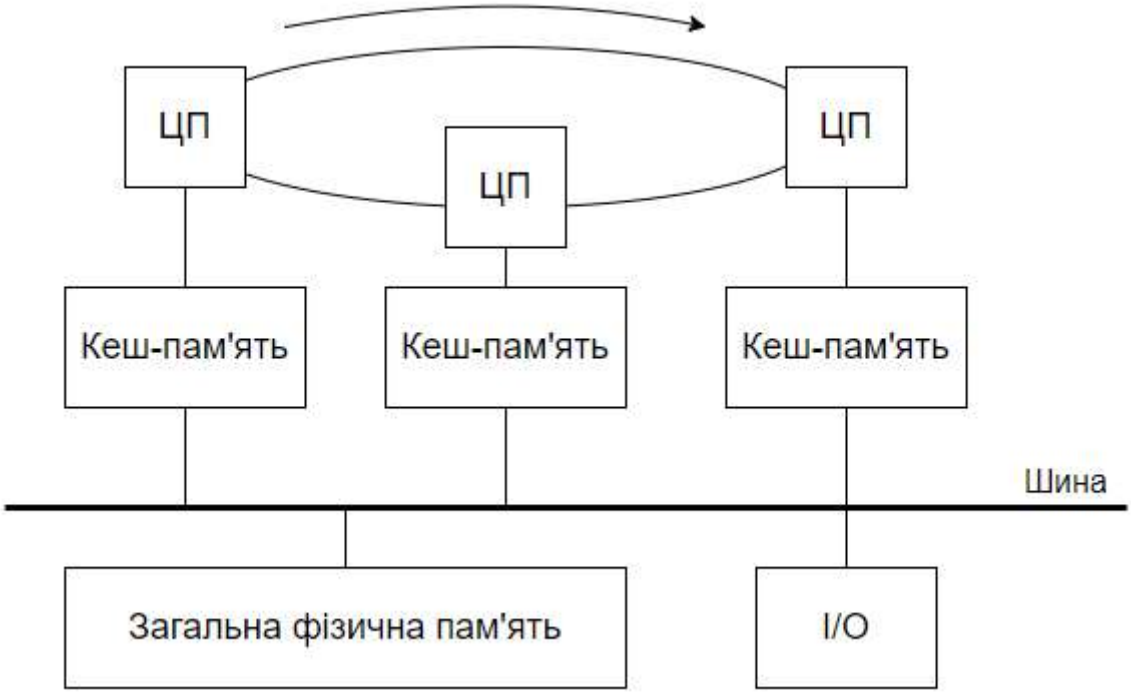


Рисунок 3.2 – Схема з’єднання елементів за допомогою топології «кільце»

Розглянувши схему можна побачити, що вона працює в напівдуплексному режимі, тобто рух здійснюється лише в одному напрямку, що не є доцільним при необхідності використання останнього (третього) компонента. Це пов’язано з тим, що для здійснення такої дії необхідно обійти усі елементи схеми, перш ніж досягнути потрібного. Для покращення такої системи варто використати

повнодуплексний режим, або в даному випадку так звану топологію подвійного кільця (рис. 3.3).

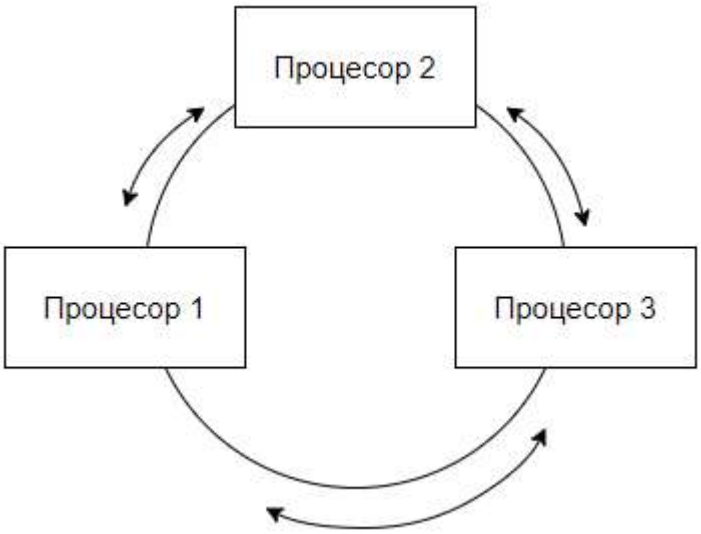


Рисунок 3.3 – Топологія подвійного кільця

Для досягнення цього необхідно забезпечити двостороннє з’єднання між компонентами (процесорами) системи. У даній мультипроцесорній системі використано шину XDBus, для з’єднання процесорів між собою, що забезпечує двонаправлену роботу без застосування додаткового з’єднання. Нею ж процесори приєднані до загальної фізичної пам’яті, що дозволяє підтримувати швидкість понад 310 Мбайт/с, при піковій продуктивності в 400 Мбайт/с. Дана шина також підтримує мультипроцесорну симетричну обробку та нарощування (масштабування) системи.

В якості шини введення/виведення застосовано SBus, у ній наявна групова пересилка даних та прямий доступ до DVMA, що дозволяє досить ефективно обмінюватись даними великих розмірів, а отже збільшувати загальну продуктивність системи.

Переваги топології подвійного кільця:

– низький ризик зіткнення даних;
– висока швидкість передачі даних;
– більша стійкість до відмов порівняно з попередньою топологією;

– вартість установки досить низька, оскільки не потрібно встановлювати додаткові компоненти;

– можливість застосування симетричної обробки даних.

Вигляд схеми після модифікації показано на рисунку 3.4.

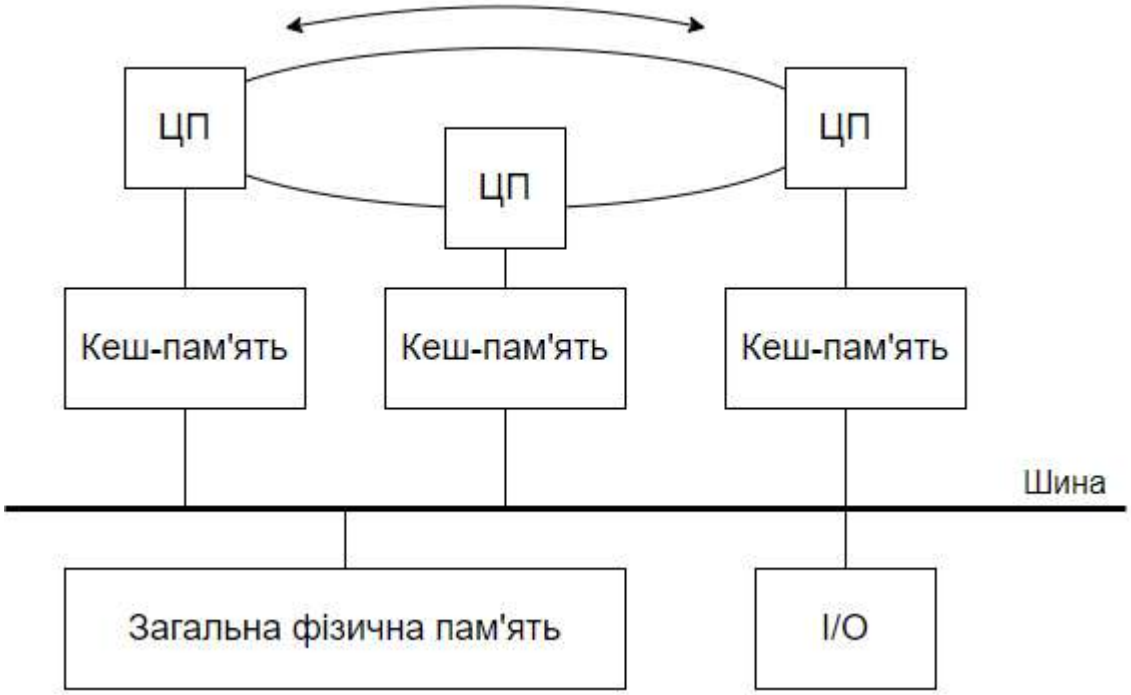


Рисунок 3.4 – Схема з’єднання елементів за допомогою топології подвійного кільця

Звичайно навіть у покращеної схеми та топології наявні свої недоліки, яких неможливо уникнути.

Недоліки такої топології:

– робота усієї системи залежить від функціонування кожного компонента;

– збільшення кількості процесорів, призводить до збільшення затримки обміну даними;

– вносити зміни в топологію можливо лише при вимкненні усієї схеми, а отже виникають простої системи;

– робота схеми залежить від функціонування компонентів зв’язку.

Загалом, покращена схема дозволяє збільшити надійність системи та зменшити її вартість. Також така система дає переваги в швидкості порівняно з схемою із застосуванням топології «кільце».

3.2 Програмна реалізація мультипроцесорної системи

Наступний крок в розробці мультипроцесорної системи є програмна реалізація. Це необхідно для паралельного розбиття програм та задач на процесори. Для цього спершу було встановлено та налаштовано UNIX-подібну операційну систему, яка підтримує багатозадачність системи. Про що свідчить поява в меню boot loader, після інсталяції. Також це означає, що було встановлено не одну, а дві версії операційної системи: з підтримкою симетричної мультипроцесорної системи та без неї. Варто зазначити, що процес встановлення ОС на багатопроцесорні системи не відрізняється від однопроцесорних. Проте, для кращого функціонування системи не достатньо лише операційної системи, потрібно використати додаткове рішення.

Як було зазначено раніше в пункті 2.5, для підтримки паралельної роботи мультипроцесорної системи варто застосовувати ще й паралельне програмування. Воно необхідне для забезпечення ефективної роботи та рівномірного розподілу задач на процесори. Для розробки даної системи використано технологію OpenMP. Оскільки технологія дозволяє виконувати програму як паралельно, так і послідовно, без зміни коду. Також, перевагою такого підходу є підтримка інкрементального програмування та цілковите керування системою, тобто створення потоків та безпосередньо керування ними.

При використанні OpenMP задіяні такі ключові елементи:

– директива parallel, для створення потоків в програмі;

– директиви do/for та section, для безпосереднього розподілу задач між процесорами;

– вирази private та shared, для контролю та керування роботи процесорів із даними;

– директиви barrier та critical atomic, для синхронізації роботи процесорів;

– процедури omp_get_thread_num та інших, для керування часом виконання;

– змінні OMP_NUM_THREADERS та інші.

У технології OpenMP використовується модель роботи «розгалуження та злиття». Тобто програма може починатись як єдиний потік, а потім розгалужуватись в потрібній області, за допомогою створення групи нових потоків (рис. 3.5). При цьому, єдиний потік стає головним у новоствореній групі, яка виконує завдання в паралельній області, що закінчується неявним бар'єром. Для розгалуження циклів в OpenMP було застосовано директиву #pragma omp for, а для блоків коду - #pragma omp sections. Приклад коду наведено в додатку А.

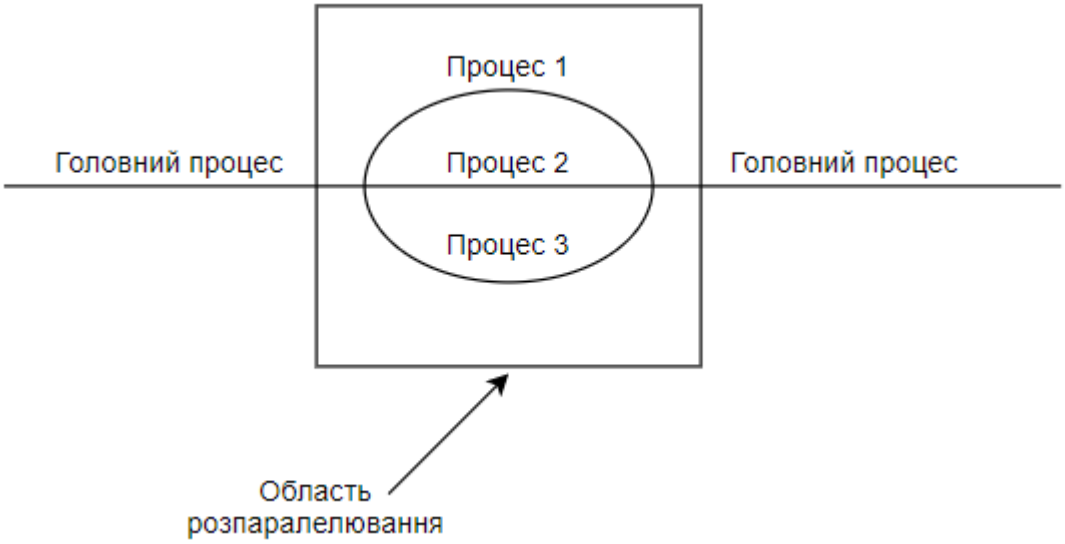


Рисунок 3.5 – Розпаралелювання програми за допомогою OpenMP

За допомогою технології OpenMP реалізовано контроль створених потоків з використанням кількох способів:

– змінної OMP_NUM_THREADS;

– процедури omp_set_num_threads();

– виразу num_threads та директиви parallel.

Це здійснено для того, щоб система мала найбільшу продуктивність, тобто кількість створених процесів була достатньою для максимальної ефективності роботи.

Для підтримки синхронізації роботи системи використано API- функції технології OpenMP. Що включаються за допомогою заголовного файла omp.h.

Значна частина коду мультипроцесорної системи реалізована з використанням директив компілятора OpenMP. Вони зазначаються з вказівками процесорів, за допомогою ключових слів #pragma omp, де omp використано для виключення збігів імен в основній програмі.

Усі директиви OpenMP, що були використані в розробці, можна розділити на три категорії:

– синхронізація – застосовуються для синхронізації роботи процесорів між собою, що дозволяє підвищити ефективність;

– визначення паралельної області – застосовуються для розмежування послідовних та паралельних областей коду;

– розподіл роботи – застосовується для визначення кількості процесорів, що будуть задіяні під час виконання, та безпосереднього розподілу задач.

Кожна із задіяних директив має свої основні та додаткові атрибути, їх ще називають опціями (clause). Зазвичай їх використовують для призначення класів змінних, які одночасно можуть виступати атрибутами для різних директив.

Для розмежування функцій паралельної програми від послідовної, використано префікс omp_, це значно покращує читабельність коду системи, що дозволяє пришвидшити його написання. Також, для коректності такої програми стосовно звичайного компілятора, підключено бібліотеку, що дозволяє встановити для кожної паралельної функції «заглушку» (stub). В обраному процесорі AMD, така бібліотека підключається за допомогою ключа –openmp-stubs.

Також, наявні ще й такі параметри компілятора AMD, для керування усіма перевагами та можливостями OpenMP:

– -openmp – застосовується для компіляції паралельного коду;

– -openmp-profile – застосовується для компіляції з інструментацією коду;

– -openmp-stubs – застосовується для ігнорування паралельного виконання програми.

При написанні програми, для мультипроцесорної системи, застосовано бібліотеки, які підтримують багатопотоковість. Вони необхідні для повноцінної роботи програми з використанням розпаралелювання.

Для відображення ефективності роботи системи написано програму, що здійснює обрахунок таких задач, із зазначенням часу:

– точковий добуток та множення матриць;

– нормалізація тривимірного вектору;

– розрахунок множини Мандельброта;

– моделювання (обчислення) N- вимірної фігури.

Код реалізації програми подано в додатку А. Результат виконання програми зображено на рисунку 3.6.

```
Mean = 4.2259 ns, StdErr = 0.0937 ns (2.22%); N = 39, StdDev = 0.5850
 ns
Min = 3.3590 ns, Q1 = 3.6899 ns, Median = 4.2062 ns, Q3 = 4.6818 ns,
Max = 5.7632 ns
IQR = 0.9919 ns, LowerFence = 2.2021 ns, UpperFence = 6.1697 ns
ConfidenceInterval = [4.0423 ns; 4.4095 ns] (CI 95%)
Skewness = 0,301627374230437, Kurtosis = 2,32560078522411


// **************************
// Benchmark: SimpleVectors.MatrixMultNaive: DefaultJob
// *** Generate ***
// Result = Success
// BinariesDirectoryPath = C:\Users\snizh\simd-workshop-master\simd-w
orkshop-master\bin\x64\Debug

// *** Build ***
BuildScript: C:\Users\snizh\simd-workshop-master\simd-workshop-master
\bin\x64\Debug\BDN.Generated.bat
// Result = Success

// *** Execute ***
// Launch: 1 / 1

// Benchmark Process Environment Information:
// Runtime=Clr 4.0.30319.42000, Arch=64-bit RELEASE [RyuJIT]
// GC=Concurrent Workstation
// Job: DefaultJob
```

Рисунок 3.6 – Результат роботи паралельної програми

На рисунку подано час та результат виконання обрахунків задач із застосуванням технології OpenMP на багатопроцесорній системі. Варто зауважити, що швидкість виконання зросла втричі, порівняно із виконанням такої ж задачі на однопроцесорній. Це свідчить про правильність застосування технології OpenMP в коді програми.

3.3 Представлення моделі роботи мультипроцесорної системи

Щоб зрозуміти як працює мультипроцесорна система, варто розглянути її роботу докладніше. Одними із головних завдань системи є міжпроцесорна взаємодія та надання доступу до системної шини. Від цього залежить ефективність та продуктивність системи.

Зважаючи на те, що пам'ять та пристрої вводу-виводу в системі спільні, необхідно забезпечити додаткову логіку, щоб не створювати проблем доступу до спільних ресурсів. Для цього необхідно забезпечити можливість доступу до шини, в деякий момент часу, лише одному процесору. Вирішенням цієї проблеми є строго визначений спосіб взаємодії компонентів (рис. 3.7).

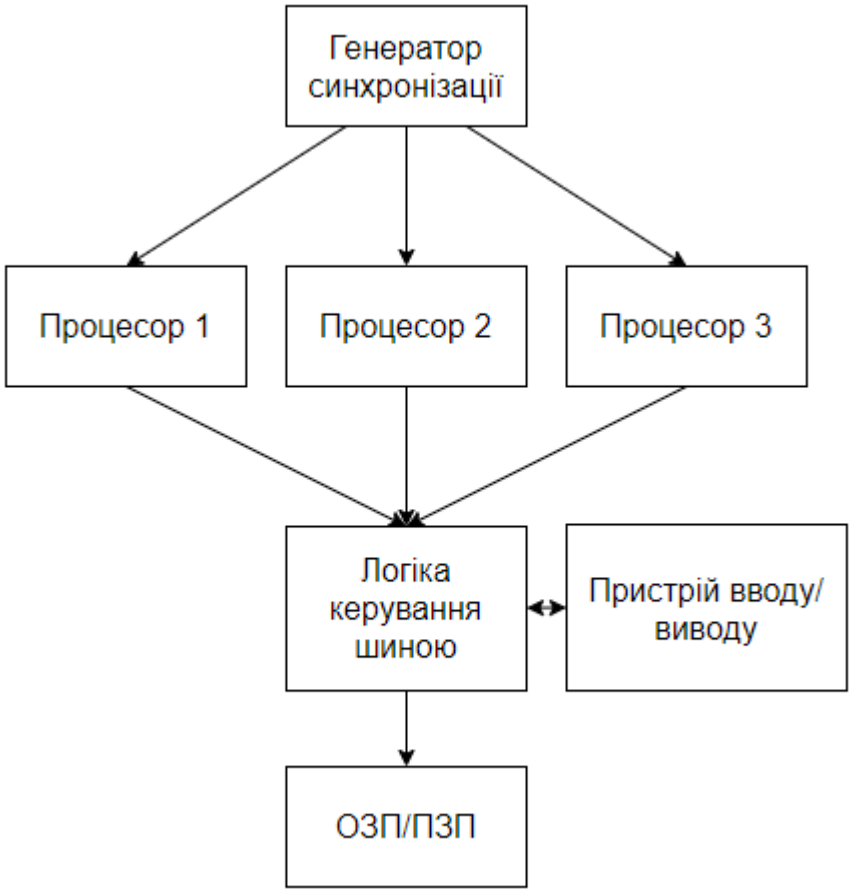


Рисунок 3.7 – Схема взаємодії мультипроцесорної системи

На рисунку 3.7 зображено схему взаємодії розроблюваної системи. У випадку такої конфігурації, усі процесори ділять не лише пам'ять та пристрій вводу/виводу, а і генератор синхронізації та логіку керування шиною, що запобігає виникненню проблеми доступу. У розроблюваному ПЗ усі компоненти рівні між собою, тому керування доступом до шини можуть здійснювати усі процесори. Що надає перевагу в швидкості порівняно з іншими подібними системами.

Також, варто розуміти, як працює планувальник завдань у розроблюваній системі (рис. 3.8). Планування - це призначення процесам пріоритет виконання в загальній черзі.

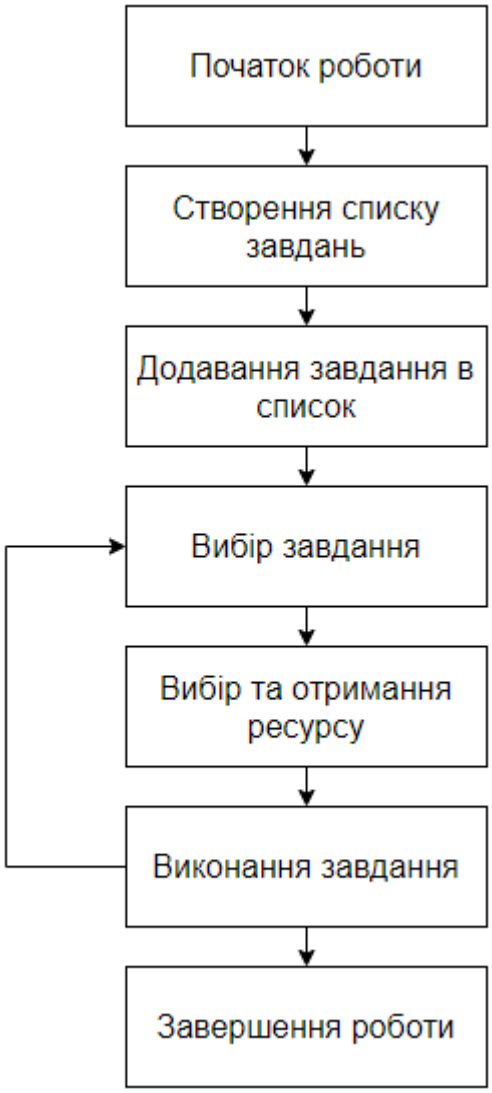


Рисунок 3.8 – Схема роботи планувальника завдань

При плануванні приймається до уваги час очікування в загальній черзі, безпосередній час виконання, пріоритет та інтенсивність звертання до периферійних пристроїв і т.д. Операційна система планує виконання потоків не приймаючи до уваги яким процесам вони належать. Що дозволяє виконання потоків різних процесів послідовно, або ж паралельно.

Загалом планування вирішує два головних завдання: визначення часу для зміни потоку та вибір наступного для виконання із загальної черги. Причому сам процес під час планування перебуває в одному із п’яти станів, що зображено на рисунку 3.9.

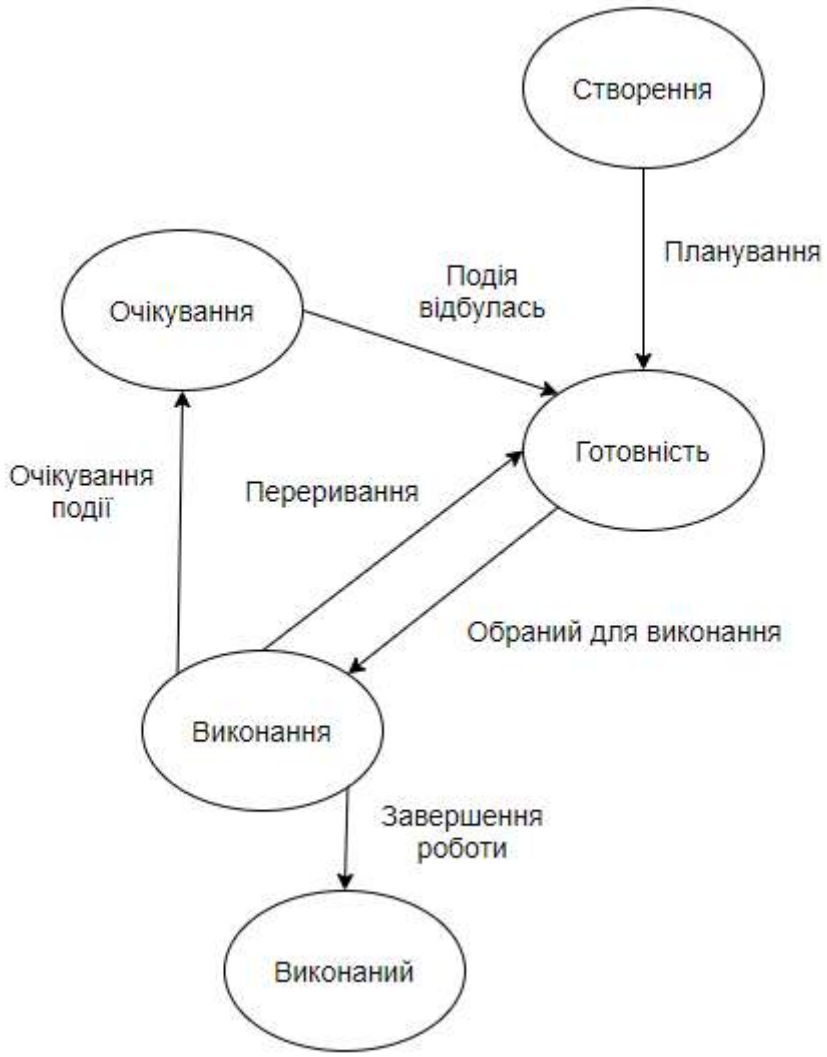


Рисунок 3.9 – Діаграма станів процесу

Кожний новий процес потрапляє в стан «створений», та отримує:

– адресний простір, для розміщення коду процесу;

– стек та системні ресурси;

– початкове значення програмного лічильника.

Після планування, створений процес потрапляє в стан готовності та очікує в черзі процесів. Далі ОС, використовуючи алгоритм планування, обирає один процес із черги та змінює його стан в «виконання». У цьому стані здійснюється безпосереднє виконання процесу, із переходом в наступний із трьох можливих станів. Вибір наступного стану здійснюється за таких трьох причин:

– або процес був цілком виконаний;

– або процес не може продовжити роботу, без настання деякої події, тому ОС змушена змінити його стан в «очікування»;

– або ОС використовує витісняючий алгоритм планування та через виникнення переривання повертає його в стан готовності.

Після закінчення виконання процесу та потрапляння його в стан виконаний, ОС обирає наступний процес для виконання. Такі дії планувальник здійснює для кожного процесора системи. Варто зазначити, що модель станів процесів для ОС UNIX складається ще з 4 проміжних станів, які використовують для більшої деталізації. Проте така модель не є доцільною для розроблюваної системи.

3.4 Представлення результатів тестування ПЗ

При розробці мультипроцесорної системи, важливо переконатись у правильності та стабільності її роботи, оскільки не можливо уникнути усіх проблем та помилок реалізації. Для цього потрібно провести тестування готового пристрою цілком та окремих його частин відповідно деяким критеріям.

Важливими критеріями оцінки мультипроцесорної системи є продуктивність та ефективність. Тому було проведено порівняння ефективності розроблюваної системи з системами, що здійснюють послідовне виконання програм, а саме однопроцесорних. Для цього було здійснено декілька випробувань із вирішенням завдань, які раніше згадувались в пункті 3.2. Причому використана технологія OpenMP дозволила не вносити зміни в код при перенесенні на однопроцесорні системи. Що є досить зручним враховуючи масштабність та важкість коду. Також, порівняння дає змогу визначити проблеми й недоліки системи та можливість їх виправлення.

Таблиця 3.1 – Порівняння ефективності роботи однопроцесорної та мультипроцесорної систем

Задача	Час виконання, нс	
	Однопроцесорна	Мультипроцесорна
Точковий добуток та множення матриць	15,8	3,6
Нормалізація тривимірного вектору	16,5	3,8
Розрахунок множини Мандельброта	18,9	4
Моделювання (обчислення) N- вимірної фігури	22,5	4,2

За результатами першого тестування, можна побачити, що розроблювана система працює значно швидше, порівняно з однопроцесорною системою. Отже вимога ефективності, поставлена в першому розділі цілком виконується. Також варто зауважити, що зменшення швидкості виконання є результатом розпаралелення завдання на три процесори. Проте досягти максимальної ефективності системи не вдалось, оскільки для її забезпечення необхідно більш масштабні задачі.

Для досягнення максимальної ефективності проведемо наступний тест, з використанням більших вхідних даних (таблиця 3.2).

Таблиця 3.2 – Результати другого тестування (для більших даних)

Задача	Час виконання, нс	
	Однопроцесорна	Мультипроцесорна
Задача 1	40,1	5,6
Задача 2	42,3	5,9
Задача 3	47	6,1
Задача 4	45,1	6

Результати другого тесту показали велику перевагу розробленої системи, а отже можна сказати, що система пройшла тестування ефективності. Також

показники систем можна порівняти більш наглядно за допомогою діаграми (рис. 3.10).

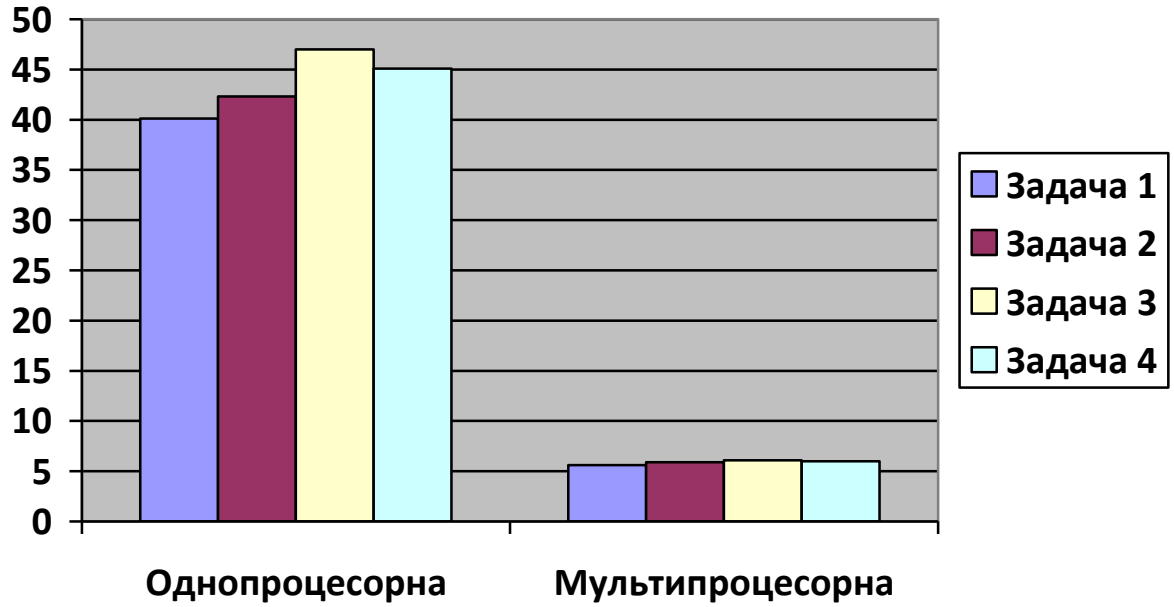


Рисунок 3.10 – Діаграма результатів другого тесту

Ще одним важливим критерієм оцінки системи є доцільність використання саме трьох процесорів для обраних задач. Для цього проведемо обчислення точкового добутку та множення матриць на системах з різною кількістю процесорів. Результати тестування подано в таблиці 3.3.

Таблиця 3.3 – Результати тестування при різній кількості процесорів

Кількість процесорів	1	2	3	4	5
Час виконання обчислення задачі 1, нс	15,8	4,7	3,6	3,6	3,6

За результатами дослідження можна побачити, що використання чотирьох чи п’яти процесорів не дало прискорення системи. Це пов’язано з тим, що для максимального розпаралелення задачі потрібно лише 3 потоки, а отже і три

процесора. Тому збільшення кількості процесорів не покращує швидкість роботи системи, але їх зменшення спричиняє збільшенню часу обрахунку.

В таблиці 3.4 подано завантаженість різної кількості процесорів при обчисленні задачі 1. Тобто, який відсоток від загального часу обчислень був задіяний процесор. Наведені дані отримані під час імітації роботи системи з пріоритетним призначення задачі процесору.

Таблиця 3.4 – Результати завантаженості процесорів

Кількість процесорів						Середня завантаженість процесора
1	100%					100%
2	63%	61%				62%
3	35%	34%	32%			33,6%
4	35%	34%	32%	0%		25,25%
5	35%	34%	32%	0%	0%	20,2%
Номер процесора	1	2	3	4	5	-

Як показано в таблиці 3.4, найбільш завантаженим процесором являється перший. Це пов’язано з тим, що у списку процесорів він знаходиться найближче, а отже після виконання завдання він одразу отримає наступне. Також можна побачити, що при надлишковій кількості процесорів в системі, усі зайві не використовуються і їх продуктивність рівна нулю. Тому для розроблюваної системи ідеальна кількість процесорів становить три.

Варто зазначити, що навантаження на перші три процесори відбувається порівняно рівномірно, що свідчить про синхронізацію роботи процесорів та використання OpenMP.

Отже, провівши тестування, стає зрозумілим, що система немає вагомих проблем які варто виправляти. Проте є речі які можна покращити, про це вказано в розділі 3.5.

3.5 Можливі способи покращення мультипроцесорної системи

Незважаючи на значні переваги системи, завжди є речі, які можна покращити, щоб система була більш надійною та швидкою. Для розроблюваної системи такими речами є архітектура та паралельне програмування.

Як раніше було зазначено в пункті 3.1, зв'язок між компонентами здійснюється за допомогою шини, проте це не дає достатньої продуктивності та швидкості системи. Тому з метою покращення, варто прибрати шину, але залишити загальний доступ до пам’яті шляхом її заміни на комутатор, або систему комутаторів (рис. 3.11).

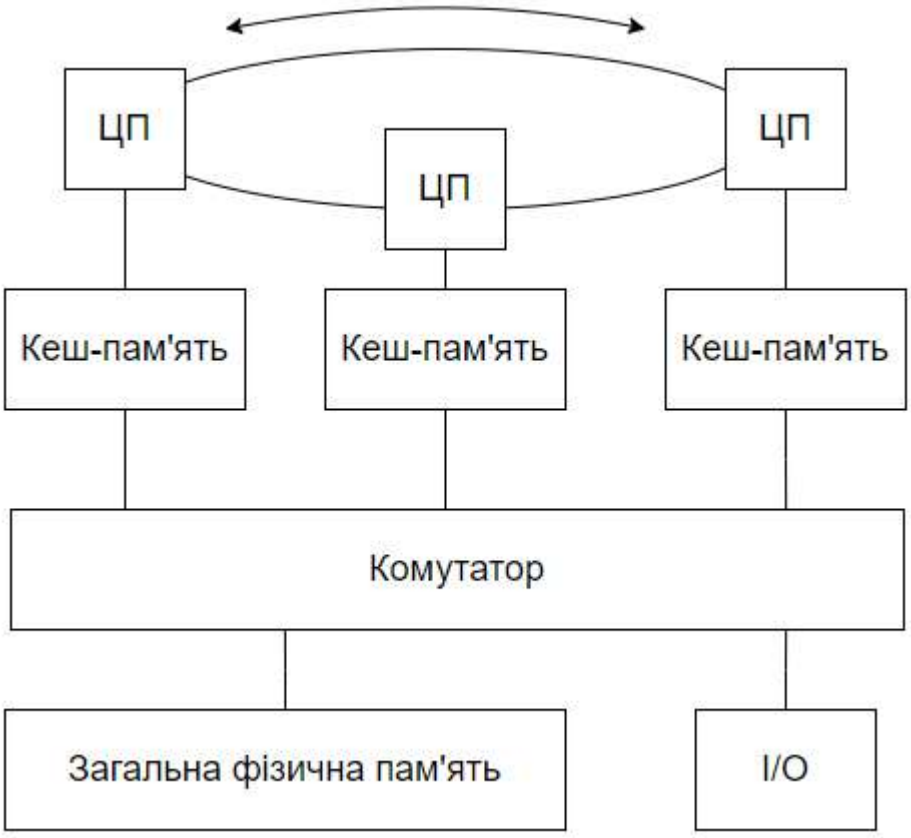


Рисунок 3.11 – Покращена архітектура системи з використанням комутатора

Така архітектура дозволяє пришвидшити роботу системи, й відповідно збільшити її загальну ефективність та продуктивність. Також вона забезпечує високу масштабованість, проте вартість такої системи значно вища відносно попередньої.

Ще одним недоліком архітектури розроблюваної системи є когерентність кешу. Для вирішення цієї проблеми можна застосувати протоколи спостереження, оскільки, для контролю стану кешу, вони застосовують вже існуючі з’єднання. Це досить зручно, адже у мультипроцесорній системі використано процесори з вбудованою кеш-пам’яттю.

У протоколах знаходиться інформація про кожну кеш-пам'ять, а саме: блок деяких даних та їх стан. Таке кешування подвоює пропускну здатність шини, але вимагає складніших апаратних засобів і сприяє виникненню затримок пересилки даних. Що може сповільнити систему, проте це не є головним недоліком.

При тестуванні в пункті 3.4, було виявлено незначний недолік паралельного програмування. Використання OpenMp не дозволило розподілити обчислення задач на більшу кількість процесорів. Щоб вирішити цю проблему варто використати OpenMp та MPI разом. Це дозволить розпаралелювати усю програму, а не лише її частини.

Для цього необхідно кожному процесору надати MPI процес, на який приходиться деяка кількість потоків OpenMp. Відповідно загальна кількість потоків збільшується втричі, а час виконання зменшується в залежності від виконуваних задач (таблиця 3.5).

Таблиця 3.5 – Порівняння часу виконання програми із застосуванням OpenMP та MPI/ OpenMP

Задача	Час виконання, нс	
	OpenMP	MPI/ OpenMP
Точковий добуток та множення матриць	3,6	2,6
Нормалізація тривимірного вектору	3,8	2,9
Розрахунок множини Мандельброта	4	3,1
Моделювання (обчислення) N- вимірної фігури	4,2	3,3

За результатами порівняння, можна побачити що використання гібридного коду дало значне пришвидшення виконання завдань. Проте таке застосування має

й свої недоліки, такі як не можливість вибору оптимальних розмірів блоку розбиття даних в середині вузла. Тому застосування гібридних технологій доцільне лише у випадку завдань, що не потребують додаткового розбиття.

3.6 Висновки

В даному розділі реалізовано мультипроцесорну систему загального призначення на основі топології «кільце». Для цього усі компоненти системи, які було обрано в пункті 2, з’єднано за допомогою шин XDBus та SBus. Використання топології, в з’єднанні, дозволило збільшити надійність та ефективність розроблюваної системи, з уникненням додаткових витрат на обладнання.

Щоб покращити розпаралелювання задач, написано програму із застосуванням технології OpenMP. Для неї було проведено тестування із застосуванням різних по важкості задач. Також було здійснено тестування системи в порівнянні з системами, що включають в себе різну кількість процесорів. На основі чого, знайдено недоліки розроблюваної системи та запропоновано способи її покращення.

ВИСНОВКИ

В результаті виконаних теоретичних та практичних досліджень було запропоновано схему побудови багатопроцесорної системи. На основі чого було розроблено мультипроцесорну систему загального призначення на основі топології «кільце».

У першому розділі було досліджено предметну область та розглянуто наявні архітектурні рішення подібних систем. В результаті чого, визначено головні вимоги до системи та здійснено постановку задачі. Такий підхід дозволив значно скоротити час розробки та зменшити загальну вартість системи.

У другому розділі обрано найбільш вдалу архітектуру побудови мультипроцесорної системи. Зважаючи на потреби системи й користувачів, було вибрано спосіб обчислювального процесу та метод розділення доступу до спільної пам’яті. Для підбору найкращих засобів, виконане порівняння операційних систем, апаратного та програмного забезпечення.

Також, в розділі було здійснено оцінку швидкості роботи багатопроцесорної системи з використанням різної кількості процесорів та із застосуванням найбільш дієвих технологій паралельного програмування. В результаті чого, було обрано технологію, що задовольняє усі вимоги системи та значно покращує продуктивність мультипроцесорної системи.

У третьому розділі було реалізовано мультипроцесорну систему. Для досягнення цього усі компоненти системи з’єднано за допомогою шин. Використання топології, в з’єднанні, дозволило збільшити надійність та ефективність розроблюваної системи, з уникненням додаткових витрат на обладнання.

Щоб покращити розпаралелювання задач, написано програму із застосуванням технології OpenMP. Також було здійснено тестування системи в порівнянні з системами, що включають в себе різну кількість процесорів. На основі чого, знайдено недоліки розроблюваної системи та запропоновано способи її покращення.

Впровадження результатів роботи дозволило реалізувати мультипроцесорну систему загального призначення на основі топології «кільце».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Agrawal, T.K., Sahu, A., Ghose, M. et al. Scheduling chained multiprocessor tasks onto large multiprocessor system. *Computing 2017*, № 99. С 1007–1028. DOI: https://doi.org/10.1007/s00607-017-0543-z

2. Romankevich, A.M., Romankevich, V.A. Diagnosis of multiprocessor systems under failure of more than half processors. *Autom Remote Control* 2017, № 78. С. 1614–1618. DOI: https://doi.org/10.1134/S0005117917090065

3. Karavay, M.F., Podlazov, V.S. An extended generalized hypercube as a fault-tolerant system area network for multiprocessor systems. *Autom Remote Control* 2015, № 76. С. 336–352. DOI: https://doi.org/10.1134/S0005117915020137

4. Lotfi, N. Ensemble of multi-objective metaheuristics for multiprocessor scheduling in heterogeneous distributed systems: a novel success-proportionate learning-based system. *SN Appl. Sci* 2019, № 1. С. 1398. DOI: https://doi.org/10.1007/s42452-019-1477-1

5. Dimitriev, Y.K. Necessary and sufficient conditions for t-diagnosability of multiprocessor computer systems for various models of nonreliable testing established using the system graph-theoretical model. *Autom Remote Control* 2016, № 77. С. 1060–1070. DOI: https://doi.org/10.1134/S0005117916060096

6. Kumar, B.K., Thanikachalam, A., Kanakasabapathi, V. et al. Performance analysis of a multiprogramming–multiprocessor retrial queueing system with orderly reattempts. *Ann Oper Res* 2016, № 247. С. 319–364. DOI: https://doi.org/10.1007/s10479-015-2005-3

7. Furugyan, M.G. Computation planning in multiprocessor real time automated control systems with an additional resource. *Autom Remote Control* 2015, № 76. С. 487–492. DOI: https://doi.org/10.1134/S0005117915030121

8. Karavai, M.F., Podlazov, V.S. Distributed full switch as an ideal system area network for multiprocessor computers. *Autom Remote Control* 2013, № 74. С. 710–724. DOI: https://doi.org/10.1134/S0005117913040140

9. Kao, CC., Lin, YC. Designs of Low Power Snoop for Multiprocessor System on Chip. *J Sign Process Syst* 2017, № 88. С. 83–89. DOI: https://doi.org/10.1007/s11265-016-1135-4

10. Furugyan, M.G. Synthesizing a Multiprocessor System for Scheduling with Interruptions and Execution Intervals. *J. Comput. Syst. Sci. Int*. 2019, № 58. С. 194–199. DOI: https://doi.org/10.1134/S1064230719020072

11. Maruf, M.A., Azim, A. Requirements-preserving design automation for multiprocessor embedded system applications. *J Ambient Intell Human Comput* 2021, № 12. С. 821–833. DOI: https://doi.org/10.1007/s12652-020-02086-9

12. Wang, HC., Woungang, I., Yao, CW. et al. Energy-efficient tasks scheduling algorithm for real-time multiprocessor embedded systems. *J Supercomput* 2012, № 62. С. 967–988. DOI: https://doi.org/10.1007/s11227-012-0771-0

13. Tabatabaee, H., Akbarzadeh-T, M.R. & Pariz, N. Dynamic task scheduling modeling in unstructured heterogeneous multiprocessor systems. J. Zhejiang Univ. – Sci, 2014. № 15. С. 423–434. DOI: https://doi.org/10.1631/jzus.C1300204

14. Pillai, A.S., Singh, K., Saravanan, V. et al. A genetic algorithm-based method for optimizing the energy consumption and performance of multiprocessor systems. *Soft Comput* 2018, № 22. С. 3271–3285. DOI: https://doi.org/10.1007/s00500-017-2789-y

15. Gavrilov, V.S., Kazennov, G.G. Method of simulation the asymmetric memory access for solving synchronization problems in multiprocessor systems. *Russ Microelectron* 2014, № 43. С. 496–500. DOI: https://doi.org/10.1134/S1063739714070087

16. Furugyan, M.G. Scheduling in Multiprocessor Systems with Additional Restrictions. *J. Comput. Syst. Sci. Int*. 2018, № 57. С. 222–229. DOI: https://doi.org/10.1134/S1064230718020077

17. Lv, F., Cui, HM., Wang, L. et al. Dynamic I/O-Aware Scheduling for Batch-Mode Applications on Chip Multiprocessor Systems of Cluster Platforms. *J. Comput. Sci. Technol*. 2014, № 29. С. 21–37. DOI: https://doi.org/10.1007/s11390-013-1409-2

18. Regnier, P., Lima, G., Massa, E. et al. Multiprocessor scheduling by reduction to uniprocessor: an original optimal approach. *Real-Time Syst* 2013, № 49. С. 436–474. DOI: https://doi.org/10.1007/s11241-012-9165-x

19. Furugyan, M.G. Some algorithms for analysis and synthesis of real-time multiprocessor computing systems. *Program Comput Soft* 2014, № 40. С. 21–27. DOI: https://doi.org/10.1134/S0361768814010034

20. Wang, H., Wang, R., Luan, Z. et al. Improving multiprocessor performance with fine-grain coherence bypass. *Sci. China Inf. Sci*, 2015. № 58. С. 1–15. DOI: https://doi.org/10.1007/s11432-014-5175-8

21. Han, S., Park, M., Piao, X. et al. A dual speed scheme for dynamic voltage scaling on real-time multiprocessor systems. *J Supercomput*, 2015. № 71. С. 574–590. DOI: https://doi.org/10.1007/s11227-014-1310-y

22. Elliott, G.A., Anderson, J.H. Globally scheduled real-time multiprocessor systems with GPUs. *Real-Time Syst*. 2012, № 48. С. 34–74. DOI: https://doi.org/10.1007/s11241-011-9140-y

23. Cannella, E., Stefanov, T.P. Energy efficient semi-partitioned scheduling for embedded multiprocessor streaming systems. *Des Autom Embed Syst*. 2016. № 20. С. 239–266. DOI: https://doi.org/10.1007/s10617-016-9176-2

24. Szustak, L., Halbiniak, K., Wyrzykowski, R. et al. Unleashing the performance of ccNUMA multiprocessor architectures in heterogeneous stencil computations. *J Supercomput*. 2019, № 75. С. 7765–7777. DOI: https://doi.org/10.1007/s11227-018-2460-0

25. Vincke, B., Elouardi, A. & Lambert, A. Real time simultaneous localization and mapping: towards low-cost multiprocessor embedded systems. *J Embedded Systems*. 2012, № 5. С. 2012. DOI: https://doi.org/10.1186/1687-3963-2012-5

26. Tehreem, A., Khawaja, S.G., Khan, A.M. et al. Multiprocessor architecture for real-time applications using mean shift clustering. *J Real-Time Image Proc*. 2019, № 16. С. 2233–2246. DOI: https://doi.org/10.1007/s11554-017-0733-0

27. Almeida, L. Guest Editorial: From Uniprocessors to Multiprocessors: Advances in Real-Time Systems. *Real-Time Syst*. 2013, № 49. С. 401–403. DOI: https://doi.org/10.1007/s11241-013-9185-1

28. Kostenetskii, P.S., Sokolinsky, L.B. Simulation of hierarchical multiprocessor database systems. *Program Comput Soft*. 2013, № 39. С. 10–24. DOI: https://doi.org/10.1134/S0361768813010040

29. Josephson, J., Ramesh, R. A novel algorithm for real time task scheduling in multiprocessor environment. *Cluster Comput*. 2019, № 22. С. 13761–13771. DOI: https://doi.org/10.1007/s10586-018-2083-5

30. Otoom, M., Paul, J.M. Workload Mode Identification for Chip Heterogeneous Multiprocessors. *Int J Parallel Prog*. 2012, № 40. С. 184–224. DOI: https://doi.org/10.1007/s10766-011-0175-4

31. Asad, A., Dorostkar, A. & Mohammadi, F. A novel power model for future heterogeneous 3D chip-multiprocessors in the dark silicon age. *J Embedded Systems*. 2018, Т. 2018, № 3. DOI: https://doi.org/10.1186/s13639-018-0086-1

32. Du, J., Wang, Y., Zhuge, Q. et al. Efficient Loop Scheduling for Chip Multiprocessors with Non-Volatile Main Memory. *J Sign Process Syst*. 2013, № 71. С. 261–273. DOI: https://doi.org/10.1007/s11265-012-0703-5

33. Thompson, C., Gould, M. & Topham, N. High Speed Cycle-Approximate Simulation of Embedded Cache-Incoherent and Coherent Chip-Multiprocessors. *Int J Parallel Prog*. 2018, № 46. С. 1247–1282. DOI: https://doi.org/10.1007/s10766-018-0566-x

34. Zhang, D., Lynch, B. & Dechev, D. Queue-Based and Adaptive Lock Algorithms for Scalable Resource Allocation on Shared-Memory Multiprocessors. *Int J Parallel Prog*. 2015, № 43. С. 721–751. DOI: https://doi.org/10.1007/s10766-014-0317-6

35. Duh, DR., Chen, CH. & Чанг, КН. Алгоритм швидкого песимістичного діагностування для узагальнених гіперкубових багатокомп’ютерних систем. *J Supercomput*. 2012, № 61. С. 605–618. DOI: https://doi.org/10.1007/s11227-011-0620-6

36. Kuo, CL., Yang, MJ., Chang, YM. et al. High diagnosability of a sequential diagnosis algorithm in hypercubes under the PMC model. *J Supercomput*. 2012, № 61. С. 1116–1134. DOI: https://doi.org/10.1007/s11227-011-0688-z

37. Elliott, G.A., Anderson, J.H. An optimal k-exclusion real-time locking protocol motivated by multi-GPU systems. *Real-Time Syst*. 2013, № 49. С. 140–170. DOI: https://doi.org/10.1007/s11241-012-9170-0

38. Garcia, V., Rico, A., Villavieja, C. et al. Adaptive Runtime-Assisted Block Prefetching on Chip-Multiprocessors. *Int J Parallel Prog*. 2017, № 45. С. 530–550. DOI: https://doi.org/10.1007/s10766-016-0431-8

39. Yang, MC. Conditional diagnosability of balanced hypercubes under the MM∗ model. *J Supercomput*. 2013, № 65. С. 1264–1278. DOI: https://doi.org/10.1007/s11227-013-0882-2

40. Fang, Z., Zhang, L., Carter, J.B. et al. Active memory controller. *J Supercomput*. 2012, № 62. С. 510–549. DOI: https://doi.org/10.1007/s11227-011-0735-9

41. Kapoor, H.K., Kanakala, P., Verma, M. et al. Design and formal verification of a hierarchical cache coherence protocol for NoC based multiprocessors. *J Supercomput*. 2013, № 65. С. 771–796. DOI: https://doi.org/10.1007/s11227-012-0865-8

42. Demaine, E.D., Ghodsi, M., Hajiaghayi, M. et al. Scheduling to minimize gaps and power consumption. *J Sched*. 2013, № 16. С. 151–160. DOI: https://doi.org/10.1007/s10951-012-0309-6

43. Smirnov, G.S., Stegailov, V.V. Efficiency of classical molecular dynamics algorithms on supercomputers. *Math Models Comput Simul*. 2016, № 8. С. 734–743. DOI: https://doi.org/10.1134/S2070048216060156

44. Mitropol'skii, Y.I. Electronic components and architecture of future supercomputers. *Russ Microelectron*. 2015, № 44. С. 139–153. DOI: https://doi.org/10.1134/S1063739715030063

45. Awasthi, M., Nellans, D., Sudan, K. et al. Managing Data Placement in Memory Systems with Multiple Memory Controllers. *Int J Parallel Prog*. 2012, № 40. С. 57–83. DOI: https://doi.org/10.1007/s10766-011-0178-1

46. Bamakhrama, M.A., Stefanov, T.P. On the hard-real-time scheduling of embedded streaming applications. *Des Autom Embed Syst*. 2013, № 17. С. 221–249. DOI: https://doi.org/10.1007/s10617-012-9086-x

47. Montaner, H., Silla, F., Fröning, H. et al. A new degree of freedom for memory allocation in clusters. *Cluster Comput*. 2012, № 15. С. 101–123. DOI: https://doi.org/10.1007/s10586-010-0150-7

48. Baklouti, M., Krichene, H. & Abid, M. Synchronous Communication-Based Many-Core SoC. *Arab J Sci Eng*. 2017, № 42. С. 845–857. DOI: https://doi.org/10.1007/s13369-016-2373-2

49. Zhao, J., Xu, C., Zhang, T. *et al.* BACH: A Bandwidth-Aware Hybrid Cache Hierarchy Design with Nonvolatile Memories. *J. Comput. Sci. Technol*. 2016, № 31. С. 20–35. DOI: https://doi.org/10.1007/s11390-016-1609-7

50. Maurya, A.K., Modi, K., Kumar, V. et al. Energy-aware scheduling using slack reclamation for cluster systems. *Cluster Comput*. 2020, № 23. С. 911–923. DOI: https://doi.org/10.1007/s10586-019-02965-7

Додаток А

(обов'язковий)

ЛІСТИНГ ПРОГРАМИ ОБЧИСЛЕННЯ ВЕКТОРИЗАЦІЇ ТОЧКОВОГО ДОБУТКУ ТА МНОЖЕННЯ МАТРИЦЬ

```
public class SimpleVectors
   {
      private const int MATRIX_SHAPE = 512;
      private readonly float[] a = new float[MATRIX_SHAPE * MATRIX_SHAPE];
      private readonly float[] b = new float[MATRIX_SHAPE * MATRIX_SHAPE];
      private readonly float[] c = new float[MATRIX_SHAPE * MATRIX_SHAPE];
      private readonly Point3[] pts = new Point3[4096];
      private readonly float[] xs = new float[4096];
      private readonly float[] ys = new float[4096];
      private readonly float[] zs = new float[4096];

      [Setup]
      public void Setup()
      {
         var rand = new Random(42);
         for (int i = 0; i < a.Length; ++i)
         {
            a[i] = b[i] = c[i] = (float)rand.NextDouble();
         }
         for (int i = 0; i < pts.Length; ++i)
         {
            xs[i] = (float)rand.NextDouble();
            ys[i] = (float)rand.NextDouble();
            zs[i] = (float)rand.NextDouble();
            pts[i] = new Point3
            {
               X = xs[i],
               Y = ys[i],
               Z = zs[i]
            };
         }
      }

      [Benchmark]
      public void AddVectors()
      {
         int length = a.Length;
         for (int i = 0; i < length; ++i)
            c[i] = a[i] + b[i];
      }

      [Benchmark]
```

```
        public void AddVectorsSimd()
        {
            int vectorLength = Vector<float>.Count;
            int remainder = a.Length % vectorLength, length = a.Length - remainder;
            for (int i = 0; i < length; i += vectorLength)
            {
                Vector<float> va = new Vector<float>(a, i);
                Vector<float> vb = new Vector<float>(b, i);
                Vector<float> vc = va + vb;
                vc.CopyTo(c, i);
            }
            for (int i = 0; i < remainder; ++i)
            {
                c[length + i] = a[length + i] + b[length + i];
            }
        }

        [Benchmark]
        public float DotProduct()
        {
            float sum = 0.0f;
            for (int i = 0; i < a.Length; ++i)
                sum += a[i] * b[i];
            return sum;
        }

        [Benchmark]
        public float DotProductSimd()
        {
            // TODO Implement this
            return 0.0f;
        }

        [Benchmark]
        public void MatrixMultNaive()
        {
            for (int i = 0; i < MATRIX_SHAPE; ++i)
            {
                for (int j = 0; j < MATRIX_SHAPE; ++j)
                {
                    for (int k = 0; k < MATRIX_SHAPE; ++k)
                    {
                        c[i * MATRIX_SHAPE + j] += a[i * MATRIX_SHAPE + k] * b[k *
MATRIX_SHAPE + j];
                    }
                }
            }
        }

        [Benchmark]
        public void MatrixMultReorg()
        {
```

```
            for (int i = 0; i < MATRIX_SHAPE; ++i)
            {
                for (int k = 0; k < MATRIX_SHAPE; ++k)
                {
                    for (int j = 0; j < MATRIX_SHAPE; ++j)
                    {
                        c[i * MATRIX_SHAPE + j] += a[i * MATRIX_SHAPE + k] * b[k *
MATRIX_SHAPE + j];
                    }
                }
            }
        }

        [Benchmark]
        public void MatrixMultSimd()
        {
            int vecSize = Vector<float>.Count;
            for (int i = 0; i < MATRIX_SHAPE; ++i)
            {
                for (int k = 0; k < MATRIX_SHAPE; ++k)
                {
                    Vector<float> va = new Vector<float>(a[i * MATRIX_SHAPE + k]);
                    for (int j = 0; j < MATRIX_SHAPE; j += vecSize)
                    {
                        Vector<float> vb = new Vector<float>(b, k * MATRIX_SHAPE + j);
                        Vector<float> vc = new Vector<float>(c, i * MATRIX_SHAPE + j);
                        vc += va * vb;
                        vc.CopyTo(c, i * MATRIX_SHAPE + j);
                    }
                }
            }
        }

        [Benchmark]
        public void VectorNorm()
        {
            for (int i = 0; i < pts.Length; ++i)
            {
                Point3 pt = pts[i];
                float norm = (float)Math.Sqrt(pt.X * pt.X + pt.Y * pt.Y + pt.Z * pt.Z);
                pt.X /= norm;
                pt.Y /= norm;
                pt.Z /= norm;
                pts[i] = pt;
            }
        }

        [Benchmark]
        public void VectorNormSimd()
        {
            // TODO Implement this
        }
```

```
}

struct Point3
{
    public float X;
    public float Y;
    public float Z;
        }
```

ЛІСТИНГ ПРОГРАМИ ОБЧИСЛЕННЯ ВЕКТОРИЗАЦІЇ МНОЖИНИ МАНДЕЛЬБРОТА

```
public class Mandelbrot
   {
      private const float MIN_X = -2;
      private const float MAX_X = 1;
      private const float MIN_Y = -1;
      private const float MAX_Y = 1;

      private const int ITERATIONS = 1000;
      private static readonly float THRESHOLD = (float)Math.Sqrt(2);

      private const int WIDTH = 40;
      private const int HEIGHT = 40;

      private int[] _fractal = new int[WIDTH * HEIGHT];

      [Benchmark]
      public void CalculateScalar()
      {
         float xStep = (MAX_X - MIN_X) / WIDTH;
         float yStep = (MAX_Y - MIN_Y) / HEIGHT;

         for (int x = 0; x < WIDTH; ++x)
         {
            for (int y = 0; y < HEIGHT; ++y)
            {
               ComplexF c = new ComplexF(MIN_X + x * xStep, MIN_Y + y *
yStep);
               ComplexF z = c;
               int iters;
               for (iters = 0; iters < ITERATIONS; ++iters)
               {
                  if (z.MagnitudeSquared >= THRESHOLD)
```

```
            {
               break;
            }
            z = z * z + c;
         }
         _fractal[y * WIDTH + x] = iters;
      }
   }
}

[Benchmark]
public void CalculateVector()
{
   float xStep = (MAX_X - MIN_X) / WIDTH;
   float yStep = (MAX_Y - MIN_Y) / HEIGHT;

   int vecSize = Vector<int>.Count;
   for (int x = 0; x < WIDTH; ++x)
   {
      // The real component in all the numbers in the inner loop is
      // going to be the same.
      Vector<float> vReal = new Vector<float>(MIN_X + x * xStep);
      for (int y = 0; y < HEIGHT; y += vecSize)
      {
         // The imaginary component has multiple separate values
         // because we're traversing through multiple values of 'y'
         // in each loop iteration.
         float[] imags = new float[vecSize];
         for (int i = 0; i < imags.Length; ++i)
         {
            imags[i] = MIN_Y + (y + i) * yStep;
         }
         Vector<float> vImag = new Vector<float>(imags);

         ComplexVF vC = new ComplexVF(vReal, vImag);
         ComplexVF vZ = vC;

         Vector<int> vIters = Vector<int>.Zero;
         Vector<int> vMaxIters = new Vector<int>(ITERATIONS);
         Vector<int> vIncrement = Vector<int>.One;
         Vector<float> vThreshold = new Vector<float>(THRESHOLD);
         do
         {
            // This is vector multiplication and addition
            vZ = vZ * vZ + vC;
```

```
                    // TODO Increment the number of iterations so far, for the elements
                    //      that haven't been phased out yet (magnitude less than
threshold).
                    vIters += Vector<int>.Zero;

                    // TODO Which vZ's have magnitude less than the threshold?
                    Vector<int> vLessThanThreshold = Vector<int>.Zero;
                    // TODO Which vIters have # iterations less than the maximum?
                    Vector<int> vLessThanMaxIters = Vector<int>.Zero;
                    // TODO Which elements should proceed to the next iteration?
                    Vector<int> vShouldContinue = Vector<int>.Zero;

                    // TODO Which vIters elements should be incremented in the next
iteration?
                    vIncrement &= Vector<int>.Zero;
                }
                while (vIncrement != Vector<int>.Zero);

                for (int i = 0; i < vecSize; ++i)
                {
                    _fractal[y * WIDTH + x + i] = vIters[i];
                }
            }
        }
    }


}
struct ComplexF
{
    private float _real;
    private float _imaginary;

    public ComplexF(float real, float imaginary)
    {
        _real = real;
        _imaginary = imaginary;
    }

    public float MagnitudeSquared
    {
        get { return _real * _real + _imaginary * _imaginary; }
    }
```

```
    public static ComplexF operator +(ComplexF a, ComplexF b)
    {
        return new ComplexF(a._real + b._real, a._imaginary + b._imaginary);
    }

    public static ComplexF operator *(ComplexF a, ComplexF b)
    {
        return new ComplexF(
            a._real * b._real - a._imaginary * b._imaginary,
            a._real * b._imaginary + a._imaginary * b._real
            );
    }
}
struct ComplexVF
{
    private Vector<float> _real;
    private Vector<float> _imaginary;

    public ComplexVF(Vector<float> real, Vector<float> imaginary)
    {
        _real = real;
        _imaginary = imaginary;
    }

    public Vector<float> MagnitudeSquared
    {
        get { return _real * _real + _imaginary * _imaginary; }
    }

    public static ComplexVF operator +(ComplexVF a, ComplexVF b)
    {
        return new ComplexVF(a._real + b._real, a._imaginary + b._imaginary);
    }

    public static ComplexVF operator *(ComplexVF a, ComplexVF b)
    {
        return new ComplexVF(
            a._real * b._real - a._imaginary * b._imaginary,
            a._real * b._imaginary + a._imaginary * b._real
            );}}
```

Додаток Б

(обов’язковий)

Результати роботи програмного забезпечення

```
MainTarget 10: 33554432 op, 318412200 ns, 9.4894 ns/op
MainTarget 11: 33554432 op, 289073600 ns, 8.6151 ns/op
MainTarget 12: 33554432 op, 291070100 ns, 8.6746 ns/op
MainTarget 13: 33554432 op, 285894600 ns, 8.5203 ns/op
MainTarget 14: 33554432 op, 292876100 ns, 8.7284 ns/op
MainTarget 15: 33554432 op, 294545100 ns, 8.7781 ns/op

Result 1: 33554432 op, 88261333.33 ns, 2.6304 ns/op
Result 2: 33554432 op, 96640733.33 ns, 2.8801 ns/op
Result 3: 33554432 op, 83025333.33 ns, 2.4743 ns/op
Result 4: 33554432 op, 82327733.33 ns, 2.4536 ns/op
Result 5: 33554432 op, 96224333.33 ns, 2.8677 ns/op
Result 6: 33554432 op, 95690033.33 ns, 2.8518 ns/op
Result 7: 33554432 op, 101536833.33 ns, 3.0260 ns/op
Result 8: 33554432 op, 89394233.33 ns, 2.6642 ns/op
Result 9: 33554432 op, 97285233.33 ns, 2.8993 ns/op
Result 10: 33554432 op, 99281733.33 ns, 2.9588 ns/op
Result 11: 33554432 op, 94106233.33 ns, 2.8046 ns/op
Result 12: 33554432 op, 101087733.33 ns, 3.0126 ns/op
Result 13: 33554432 op, 102756733.33 ns, 3.0624 ns/op


Mean = 2.8143 ns, StdErr = 0.0557 ns (1.98%); N = 13, StdDev = 0.2010 ns
Min = 2.4536 ns, Q1 = 2.6473 ns, Median = 2.8677 ns, Q3 = 2.9857 ns, Max = 3.0624 ns
IQR = 0.3385 ns, LowerFence = 2.1396 ns, UpperFence = 3.4934 ns
ConfidenceInterval = [2.7051 ns; 2.9235 ns] (CI 95%)
Skewness = -0,560655129210459, Kurtosis = 1,87765836373747
```

Рисунок Б.1 – Результат обчислення векторизації точкового добутку та множення матриць

```
Result 54: 4096 op, 230127553.33 ns, 56.1835 us/op
Result 55: 4096 op, 225332853.33 ns, 55.0129 us/op
Result 56: 4096 op, 235741653.33 ns, 57.5541 us/op
Result 57: 4096 op, 247792153.33 ns, 60.4961 us/op
Result 58: 4096 op, 243289553.33 ns, 59.3969 us/op
Result 59: 4096 op, 246898353.33 ns, 60.2779 us/op
Result 60: 4096 op, 232550853.33 ns, 56.7751 us/op
Result 61: 4096 op, 232261553.33 ns, 56.7045 us/op
Result 62: 4096 op, 225271053.33 ns, 54.9978 us/op
Result 63: 4096 op, 220714653.33 ns, 53.8854 us/op
Result 64: 4096 op, 236571153.33 ns, 57.7566 us/op
Result 65: 4096 op, 245475953.33 ns, 59.9307 us/op
Result 66: 4096 op, 259912253.33 ns, 63.4551 us/op


Mean = 61.8775 us, StdErr = 0.6160 us (1%); N = 66, StdDev = 5.0048 us
Min = 53.2906 us, Q1 = 57.7166 us, Median = 61.3586 us, Q3 = 65.3167 us, Max = 73.7335 us
IQR = 7.6001 us, LowerFence = 46.3164 us, UpperFence = 76.7168 us
ConfidenceInterval = [60.6701 us; 63.0850 us] (CI 95%)
Skewness = 0,353945584468786, Kurtosis = 2,37987703780158
```

Рисунок Б.2 – Результат обчислення векторизації множини Мандельброта

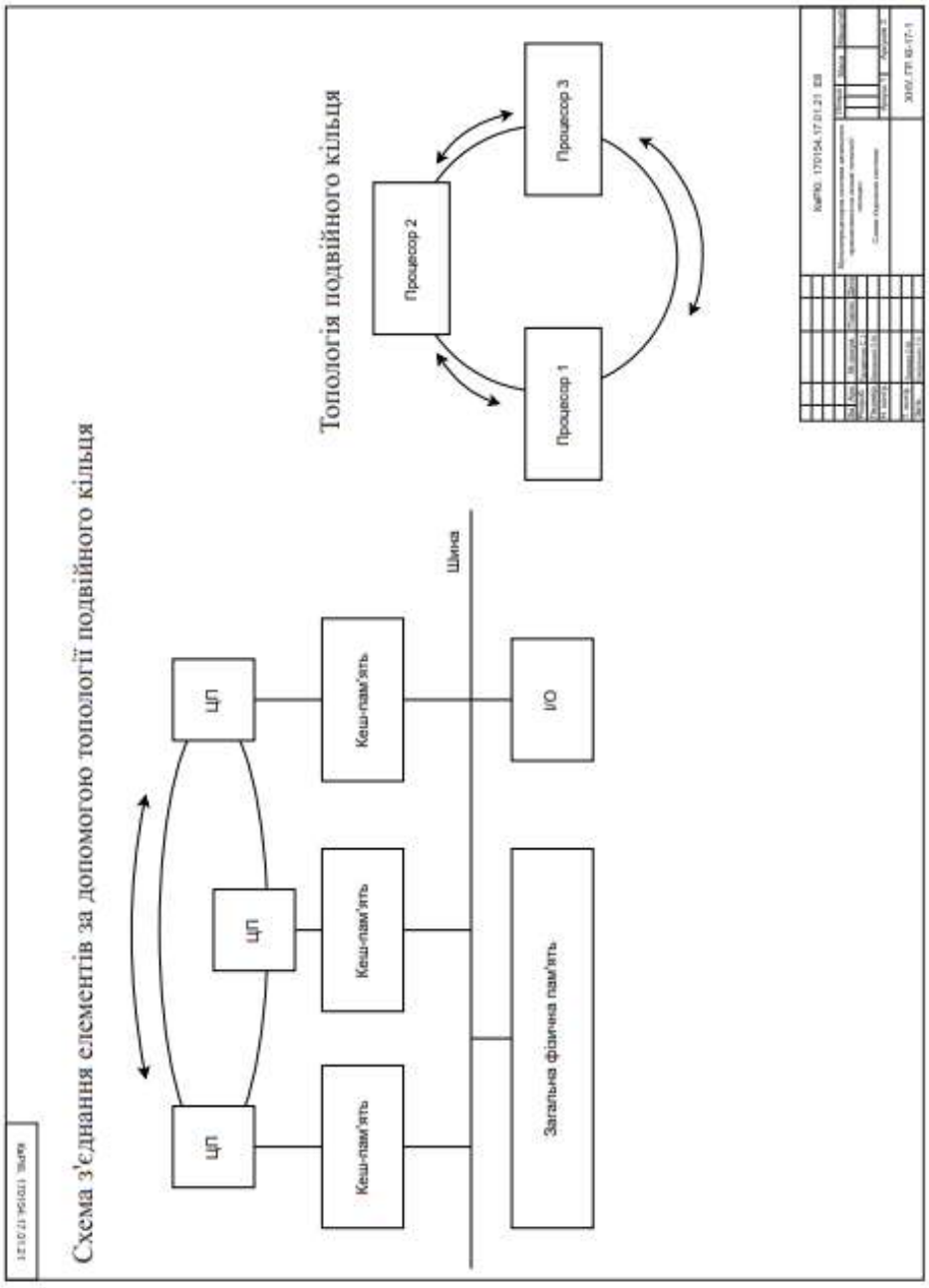
```
// * Detailed results *
Particles.Simulate: DefaultJob
Mean = 16.6853 ms, StdErr = 0.1668 ms (1%); N = 61, StdDev = 1.3030 ms
Min = 15.3272 ms, Q1 = 15.5359 ms, Median = 16.3122 ms, Q3 = 17.7003 ms, Max = 20.1762 ms
IQR = 2.1644 ms, LowerFence = 12.2893 ms, UpperFence = 20.9469 ms
ConfidenceInterval = [16.3583 ms; 17.0123 ms] (CI 95%)
Skewness = 0,897283511536162, Kurtosis = 2,74316257085404


Particles.SimulateSimd: DefaultJob
Mean = 0.8951 ns, StdErr = 0.0140 ns (1.56%); N = 14, StdDev = 0.0523 ns
Min = 0.8266 ns, Q1 = 0.8459 ns, Median = 0.8899 ns, Q3 = 0.9307 ns, Max = 1.0066 ns
IQR = 0.0849 ns, LowerFence = 0.7186 ns, UpperFence = 1.0580 ns
ConfidenceInterval = [0.8677 ns; 0.9226 ns] (CI 95%)
Skewness = 0,447285365779589, Kurtosis = 2,07722488779755
```

Рисунок Б.3 – Результат обчислення векторизації векторної нормалізації

Додаток В

(обов’язковий)

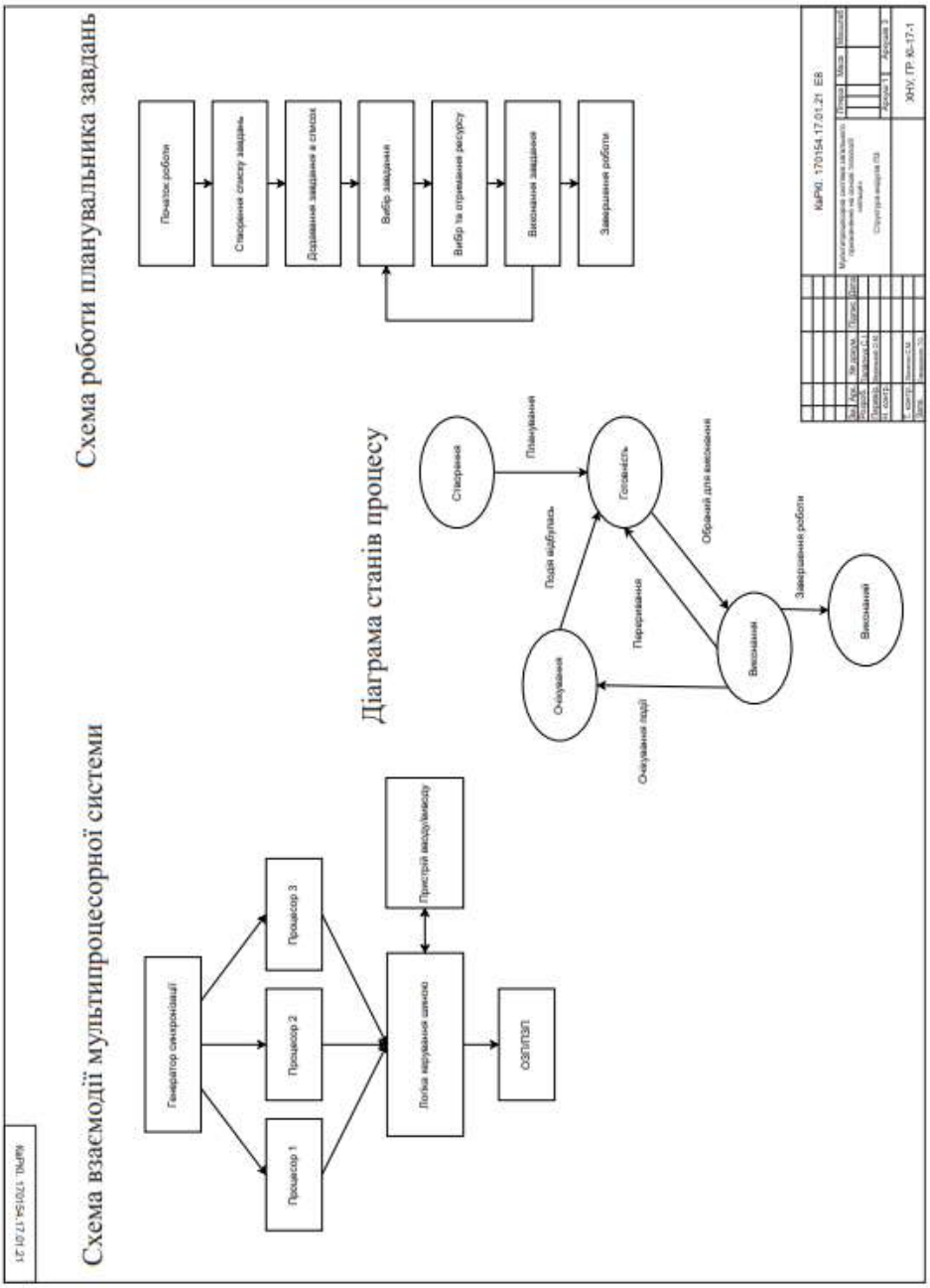
Копія креслення «Схема з’єднання системи»



Додаток Г

(обов’язковий)

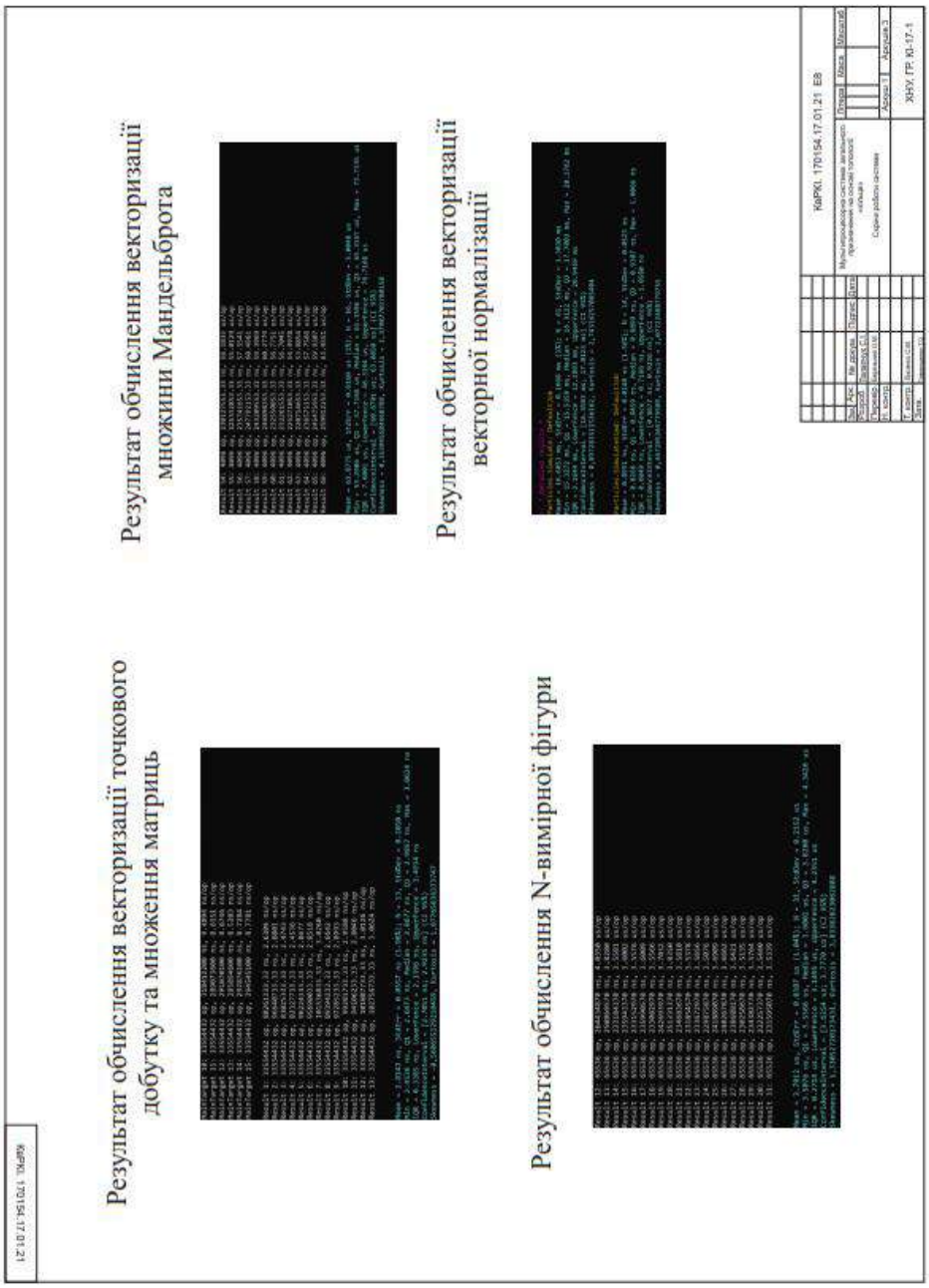
Копія креслення «Структура модулів ПЗ»



Додаток Д

(обов’язковий)

Копія креслення «Скріни роботи системи»



UNICHECK

Ім'я користувача:
Кафедра КІ

ID перевірки:
1008133534

Дата перевірки:
02.06.2021 08:21:05 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
02.06.2021 08:21:43 EEST

ID користувача:
100005591

Назва документа: **Талапчук_Мультипроцесорна система загального призначення на основі топології «кільце»**

Кількість сторінок: **71** Кількість слів: **13795** Кількість символів: **103757** Розмір файлу: **870.00 KB** ID файлу: **1008214710**

13.6%
Схожість

Найбільша схожість: **8.37%** з джерелом з Бібліотеки (ID файлу: **1008214695**)

10.9% Джерела з Інтернету	109	Сторінка 73
8.52% Джерела з Бібліотеки	59	Сторінка 75

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Anti-Plagiarism v-15.257

Максимальна збіг з одним документом 0.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 16%**

ID: 91878 Назва: Мультіпроцесорна система загально призначення на основі топології «кільце» Додано в БД: 2021-06-02 Автори: С.І. Талапчук Керівники: О.М. Березький Консультанти: опоненти:	документ		Сумарне збіг по Базі Даних	
	символи	лексеми	символи	лексеми
	82464	780	484 (1%)	10 (1%)

джерело плагіату

ID	опис	Наявність плагіату в документі	
		символи	лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНА РОБОТА

Дипломник Талапчук Сніжана Іванівна
Тема Мультипроцесорна система загального призначення на основі топології «кільце»
Спеціальність 123 Комп'ютерна інженерія

Обсяг кваліфікаційного проекту:

кількість листів креслень 3 : кількість сторінок записки 84

1. Короткий зміст КП та прийнятих рішень В рамках кваліфікаційної роботи було розроблено мультипроцесорну систему загального призначення на основі топології «кільце».

2. Висновок про відповідність КП кваліфікаційному завданню Кваліфікаційна робота у повній мірі відповідає поставленому завданню як в теоретичній, так і в практичній частині даного проекту

3. Характеристика виконання кожного розділу проекту, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому, теоретичному, розділі кваліфікаційної роботи якісно та в повній мірі розглянуті методи вирішення поставленої задачі, був проаналізований кожен аспект, який стосується теми кваліфікаційної роботи. У наступному розділі було здійснено обгрунтування обраної структури системи на основі порівняння різних можливих варіантів побудови цієї системи. У основній проектній частині роботи була реалізована сучасними методами та рішеннями мультипроцесорна система. З'єднання компонентів системи здійснено на основі новітніх можливостей із застосуванням топології «кільце». Також було здійснено налаштування мультипроцесорної системи у відповідному розділі кваліфікаційної роботи. В загальному усі розділи відповідають завданню та містять сучасні методи вирішення поставлених завдань.

4. Позитивні сторони проекту Кваліфікаційна робота відповідає сучасним вимогам до проектування мультипроцесорних систем та містить ряд інноваційних рішень, зокрема, з точки зору використання топології для з'єднання компонентів. Окремо можна виділити розглянуту можливість збільшення продуктивності системи шляхом легкого масштабування, оскільки на сьогоднішній день таке рішення є досить затребуваним і має значні перспективи. Також важливим позитивним аспектом кваліфікаційної роботи є реалізована можливість керування розподілом задач на процесори, що надає можливість користувачам повністю керувати системою.

5. Негативні сторони проекту Надмірна кількість теоретичного матеріалу, відсутність початкових налаштувань системи. В рамках кваліфікаційної роботи варто було приділити більшу увагу високошвидкісним способам обміну даними в системі, оскільки потреба в даному рішенні щороку зростає.

6. Оцінка графічного оформлення та пояснювальної записки проекту Графічне оформлення виконане відповідно до суті кваліфікаційної роботи. У першому листі креслення відображена схема з'єднання компонентів за допомогою топології. У другому листі креслення розглянуті питання модулів роботи системи. Останній із листів креслення присвячений результатам роботи мультипроцесорної системи. В загальному графічне оформлення виконане на належному рівні, єдиним недоліком є нерівномірність розмірів шрифтів, що робить розгляд графічних креслень дискомфортним. Пояснювальна записка відповідає задекларованим нормам для її оформлення.

7. Відгук про проект в цілому В загальному дипломний проект заслуговує схвальних відгуків. Весь матеріал дипломного проекту структурований, чіткий та послідовний. Усі розділи проекту йдуть у вірній послідовності, що дозволяє чітко розуміти викладений матеріал в рамках даного дипломного проекту. Графічний матеріал дозволяє наочно побачити доцільність та ефективність рішень, які були прийняті за основу при проектуванні мультипроцесорної системи.

8. Інші зауваження _____

9. Оцінка кваліфікаційної роботи. За результатами розгляду позитивних та негативних сторін представленого дипломного проекту, можна зробити висновок, що він заслуговує оцінку «відмінно»/5,0/A.

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи) доцент кафедри автоматизації, комп'ютерно-інтеграційних технологій та телекомунікаційній, к.т.н. Федула Микола Васильович

« _____ » _____ 2021 р. _____ (підпис)

Завідувачу кафедри КІСП
д-р.техн.наук, проф. Говорущенко Т. О.

Талапчук С.І.

ПІБ здобувача вищої освіти

ФПКТС, 4 курсу, групи КІ-17-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіатоповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

04.06.21

дата

підпис

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА СИСТЕМНОГО ПРОГРАМУВАННЯ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Мультипроцесорна система загального призначення на основі топології «гіперкуб»

Автор: Талапчук Сніжана Іванівна

Спеціальність: 123 – Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Березький О.М., д.т.н, професор

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні;
3) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з більш ніж 10 джерелами на один фрагмент речення;
4) серед запозичень знаходяться загальновідомі терміни, скорочення та визначення.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 13,6 і адресується до 109 першоджерела, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ О.М. Березький

Гарант ОП _____ С.М. Лисенко

Завідувач кафедри КІСП _____ Т. О. Говорущенко